Generating Known Unknowns through Known Knowns

Synthetic Adversarial Log Objects

Hi, I'm Marcus LaFerrera (@mlaferrera)

- Staff Security Strategist @ Splunk
- Previously supported many things DoD, most recently @ DARPA
- Avid open-source contributor, mostly python
- Former lead developer of stoQ
- Dreams about **automating** everything



first... some background



Leveraging Splunk's network defense and incident response techniques to defend against Sunburst Backdoor

Read the Blog >



Detecting Supernova Malware: SolarWinds

How to detect Supernova malware that uses a zero-day vulnerability to install a trojanized .NET DLL

Read the Blog >



A Golden SAML Journey: SolarWinds

How to detect and mitigate SAML attacks

Read the Blog >

THREAT RESEARCH

Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor

FIREEYE

DEC 13, 2020 | 17 MINS READ



Detecting Supply Chain Attacks

Using Splunk and JA3/s hashes to detect malicious activity on critical servers

Read the White Paper >





There's a (better) way



https://github.com/splunk/salo

Caveat: This is a work in progress

Synthetic Adversarial Log Objects (SALO) is a framework for the generation of log events without the need for infrastructure or actions to initiate the event that causes a log event. The purpose of this framework is to allow security practitioners, data scientists, and researchers the ability to create log events in a simple, repeatable, and randomized way without the overhead of traditional required resources.

But, why another fake log generator?

Low barrier to entry

Minimal effort

Repeatable process

Highly customizable

Let's walk through a simple scenario

1. Create a new recipe, dns.yaml



2. Run salo, and generate our data

\$ salo recipe dns.yaml

```
d8b
                   88P
                  d88
                  888
 .d888b, d888b8b
                        d8888b
 ?8b,
       d8P' ?88
                  ?88
                       d8P' ?88
   `?8b 88b ,88b 88b 88b d88
 ?888P' `?88P'`88b
                    88b`?8888P'
                    v0.1.0
[*] Generating synthetic events from dns.yaml...
[*] Generated 1 events.
  "ts": "2021-11-30T16:15:08Z",
  "uid": "C5niFgYVbXt8U7Juhs",
  "id.orig h": "192.168.4.82",
  "id.orig p": 49428,
  "id.resp_h": "58.54.106.99",
  "id.resp_p": 53,
  "proto": "tcp",
  "trans_id": 22900,
  "rtt": 1.31913366104463,
  "query": "web-11.hughes.com",
  "gclass": 1,
  "qtype": 28,
  "rcode": 2.
  "rcode_name": "SERVFAIL",
  "AA": true,
  "TC": false,
  "RD": false,
  "RA": false,
  "Z": 0,
  "rejected": false
```

- 3. Define a domain to query
- 4. Spawn a Zeek conn event



5. Run salo again

\$ salo recipe dns.yaml

or...

\$ salo recipe -o output.yaml --splunk dns.yaml

or...

\$ salo recipe -o output.yaml --file dns.yaml

<pre>v0.1.0 *] Generating synthetic events from dns.yaml *] Generated 2 events. "ts": "2021-11-30T16:23:262", "uid": "CJ4KmyfJPdPLYflLCV", "id.orig_h": "10.54.138.33", "id.orig_p": 58705, "id.resp_p": 53, "proto": "udp", "trans_id": 13962, "ntt": 0.72964708148341, "query": "deftsecurity.com", "qclass": 1, "qclass.name": "C_INTERNET", "qtype": 16, "qtype_name": "TXT", "rcode_name": "NXDOMAIN", "AA*: true, "tc": false, "RD": false, "RA*: false, "RA*: false, "AA*: false, "XA*: false, "XA*: false, "XA*: false, "AA*: false, "AA*: false, "AA*: false, "AA*: false, "AA*: false, "AA*: false," "id.orig_h": "221.112.84.142", "id.orig_h": "221.112.84.142", "id.resp_h": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 1938, "con_state": "RSTO", "mistory": "ShADTadtff", "orig_ptxs": 9844, "orig_ptxs": 9844, "orig_ptxs": 7628, "some tabeteent 2566</pre>	?	88P d888b, d888b8b 888 d8888b 8b, d8P' ?88 788 d8P' ?88 `?8b 88b ,88b 88b 88b d88 888P' `?88P' `88b 88b `?8888P'
<pre>*] Generating synthetic events from dns.yaml *] Generated 2 events. "ts": "2021-11-30T16:23:262", "id.": "CJ4KmyfJP4PLXfLCV", "id.orig_h": "10.54.138.33", "id.orig_p": 58705, "id.resp_h": "2011.112.84.142", "id.resp_p": 53, "proto": "udp", "trans_id": 13962, "rtt": 0.72964708148341, "query": "deftsecurity.com", "qclass_name": "C_INTERNET", "trype": 16, "qtype_name": "TXT", "rcode": 3, "rcode_name": "NXDOMAIN", "AA": true, "TC": false, "RA": false, "RA": false, "RA": false, "RA": false, "RA": false, "RA": false, "RA": false, "rejected": true "ts": "2021-11-30T16:23:272", "id.orig_h": "10.54.138.33", "id.orig_p": 58705, "id.resp_h": "221.112.84.142", "id.resp_p": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 9902, "resp_bytes": 1938, "con_state": "RSTO", "missed_bytes": 7040, "history": "ShADTadtff", "orig_pkts": 9844, "orig_pkts": 9844, "orig_bytes": 7028, "resp_bytes": 1939, "resp_bytes": 1939, "resp_bytes": 7628, "resp_bytes": 7628,</pre>		v0.1.0
<pre>"ts": "2021-11-30T16:23:262", "uid": "CJ4KmyfJP4PLXfLLCV", "id.orig_p": 58705, "id.resp_p": 537, "proto": "udp", "trans_id": J3962, "rtt": 0.72964708148341, "query": "deftsecurity.com", "qclass": 1, "qclass": 1, "qclass": 1, "qclass.name": "C_INTERNET", "qtype": 16, "ttype_name": "TXT", "rcode_name": "NXDOMAIN", "AA": true, "TC": false, "RA": false, "RA": false, "RA": false, "z": 0, "rejected": true "ts": "2021-11-30T16:23:272", "uid.orig_p": 58705, "id.orig_p": 53, "id.orig_p": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 1938, "con_state": "RSTO", "missory": "ShADTadtFF", "orig_pkts": 9844, "orig_pkts": 9264, "septem: "Secon", Secon", "septem: "secon", Secon", "setor": "udp", "septem: "ShADTadtFF", "orig_pkts": 9139, "resp_pkts": 7628, "septem: "secon", Secon", "septem: "secon", Secon", "secon", Secon", Secon", "secon", Secon", Secon", "secon", Secon", Secon", "secon", Secon", Secon", "secon", Secon", Secon", "secon", Secon", Secon",</pre>	* *] Generating synthetic events from dns.yaml.] Generated 2 events.
"Z": 0, "rejected": true "ts": "2021-11-30T16:23:27Z", "uid": "CJ4KmyfJPdPLXflLCv", "id.orig_p": 58705, "id.resp_p": 53, "id.resp_p": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 9902, "resp_bytes": 1938, "conn_state": "RSTO", "missed_bytes": 7040, "history": "ShADTadtFF", "orig_pkts": 9844, "orig_ip_bytes": 9139, "resp_hts": 7628, "proto: phytes": 2566		"ts": "2021-11-30T16:23:262", "uid": "CJ4KmyfJPdPLXflLCv", "id.orig_h": "10.54.138.33", "id.orig_p": 58705, "id.resp_h": "221.112.84.142", "id.resp_h": "221.112.84.142", "trans_id": 13962, "trans_id": 13962, "trt": 0.72964708148341, "query": "deftsecurity.com", "qclass": 1, "qclass_name": "C_INTERNET", "qtype": 16, "qtype_name": "TXT", "rcode": 3, "rcode": 3, "rcode": 41, "alse, "R0": false, "RA": false,
"ts": "2021-11-30T16:23:27Z", "uid": "CJ4KmyfJPdPLXflLCv", "id.orig_p": 58705, "id.resp_p": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 9902, "resp_bytes": 1938, "conn_state": "RSTO", "missed_bytes": 7040, "history": "ShADTadtFF", "orig_pkts": 9844, "orig_ip_bytes": 9139, "resp_bts": 7628, "prop. ab bytes": 2566		"rejected": true
(bbb		"ts": "2021-11-30T16:23:27Z", "uid": "CJ4KmyfJPdPLXflLCV", "id.orig_p": 58705, "id.orig_p": 58705, "id.resp_h": "221.112.84.142", "id.resp_p": 53, "proto": "udp", "service": "dns", "duration": 1.30902483955984, "orig_bytes": 9902, "resp_bytes": 1938, "con_state": "RSTO", "missed_bytes": 7040, "history": "ShADTadtfF", "orig_ip_bytes": 9139, "resp_pkts": 7628, "resp_bkts": 7628, "and the state": "State the state the s

Recipes | Stencils | Events | Outputs





define a scenario

scenario: Random Beacon author: Marcus LaFerrera date: 2021-09-25 description: Generate a random host resolving a domain and then beaconing over TLS options: src_ip: 192.168.54.36 time: jitter_min: 0 jitter_max: 1 sessions: - event: salo.events.zeek.ConnModel description: Generate initial DNS connection and query repeat: 1 options: dest_port: 53 proto: udp service: dns spawns: - event: salo.events.zeek.DNSModel likelihood: 100 save_values: first_dns_response: dns_rdata first_dns_query: dns_query query_timestamp: timestamp options: dest_port: 53 dns_rcode: 0 dns_rcode_name: NOERROR dns_qtype: 1 dns_qtype_name: A spawns: - event: salo.events.suricata.dns.DNSModel time: options: timestamp: \$query_timestamp dns_type: answer

scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
src_ip: 192.168.54.36
time:
jitter_min: 0
jitter_max: 1
cadence: "* 10 1 * * *"
sessions:
– event: salo.events.zeek.ConnModel
description: Generate initial DNS connection and query
repeat: 1
options:
dest_port: 53
proto: udp
service: dns
spawns:
- event: salo.events.zeek.DNSModel
likelihood: 100
save_values:
first_dns_response: dns_rdata
first_dns_query: dns_query
query_timestamp: timestamp
options:
dest_port: 53
dns_rcode: 0
dns_rcode_name: NOERROR
dns_qtype: 1
dns_qtype_name: A
spawns:
– event: salo.events.suricata.dns.DNSModel
time:
options:
timestamp: \$query_timestamp
dns_type: answer

event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
<pre>dest_ip: \$first_dns_response.0</pre>
proto: tcp
dest_port: 443
service: http,ssl
spawns:
– event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
<pre>server_name: \$first_dns_query</pre>



Global Options

event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
<pre>dest_ip: \$first_dns_response.0</pre>
proto: tcp
dest_port: 443
service: http,ssl
spawns:
– event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
<pre>server_name: \$first_dns_query</pre>

scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
src_ip: 192.168.54.36
time:
jitter_min: 0
jitter_max: 1
cadence: "* 10 1 * * *"
sessions:
– event: salo.events.zeek.ConnModel
description: Generate initial DNS connection and query
repeat: 1
options:
dest_port: 53
proto: udp
service: dns
spawns:
- event: salo.events.zeek.DNSModel
likelihood: 100
save_values:
first_dns_response: dns_rdata
first_dns_query: dns_query
query_timestamp: timestamp
options:
dest_port: 53
dns_rcode: 0
dns_rcode_name: NOERROR
dns_qtype: 1
dns_qtype_name: A
spawns:
– event: salo.events.suricata.dns.DNSModel
time:
options:
timestamp: \$query_timestamp
dns_type: answer

Timestamps, Jitter, and Cadence

– event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
<pre>dest_ip: \$first_dns_response.0</pre>
proto: tcp
dest_port: 443
service: http,ssl
spawns:
– event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
server_name: \$first_dns_query

scenario: Random Beacon	
author: Marcus LaFerrera	
date: 2021-09-25	
description: Generate a random host resolving a domain and then beacon	ing over TLS
options:	
src_ip: 192.168.54.36	
time:	
jitter_min: 0	
jitter_max: 1	
# cadence: "* 10 1 * * *"	
sessions:	
- event: salo.events.zeek.ConnModel	
description: Generate initial DNS connection and query	
repeat: 1	
options:	
dest_port: 53	
proto: udp	
service: dns	
spawns:	
– event: salo.events.zeek.DNSModel	
likelihood: 100	
save_values:	
tirst_dns_response: dns_rdata	
tirst_ans_query: ans_query	
query_timestamp: timestamp	
options:	
dest_port: 53	
dns_rcode_name: NOEDBOD	
dis_rcode_liame. NULINON	
dns ature name: A	
- event: salo events suricata dos DNSModel	
time:	
options:	
timestamp: \$query timestamp	
dns type; answer	

Sessions and Events

- event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
<pre>dest_ip: \$first_dns_response.0</pre>
proto: tcp
dest_port: 443
service: http,ssl
spawns:
– event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
<pre>server_name: \$first_dns_query</pre>

```
scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
  src ip: 192.168.54.36
time:
  jitter_min: 0
  jitter_max: 1
  # cadence: "* 10 1 * * *"
sessions:
  - event: salo.events.zeek.ConnModel
    description: Generate initial DNS connection and query
    repeat: 1
    options:
      dest_port: 53
      proto: udp
      service: dns
    spawns:
      - event: salo.events.zeek.DNSModel
        likelihood: 100
        save_values:
          first_dns_response: dns_rdata
          first_dns_query: dns_query
          query_timestamp: timestamp
        options:
          dest port: 53
          dns_rcode: 0
          dns_rcode_name: NOERROR
          dns_qtype: 1
          dns_qtype_name: A
        spawns:
          - event: salo.events.suricata.dns.DNSModel
            time:
            options:
              timestamp: $query_timestamp
              dns_type: answer
```

Repeat Events

event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
<pre>dest_ip: \$first_dns_response.0</pre>
proto: tcp
dest_port: 443
service: http,ssl
spawns:
– event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
<pre>server_name: \$first_dns_query</pre>

```
scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
  src ip: 192.168.54.36
time:
  jitter_min: 0
  jitter_max: 1
  # cadence: "* 10 1 * * *"
sessions:
  – event: salo.events.zeek.ConnModel
    description: Generate initial DNS connection and query
    repeat: 1
    options:
      dest_port: 53
      proto: udp
      service: dns
    spawns:
      - event: salo.events.zeek.DNSModel
        likelihood: 100
        save_values:
          first_dns_response: dns_rdata
          first_dns_query: dns_query
          query_timestamp: timestamp
        options:
          dest port: 53
          dns_rcode: 0
          dns_rcode_name: NOERROR
          dns_qtype: 1
          dns_qtype_name: A
        spawns:
          - event: salo.events.suricata.dns.DNSModel
            time:
            options:
              timestamp: $query_timestamp
              dns_type: answer
```

Event Options

– event: salo.events.zeek.ConnModel	
description: Initiate initial TLS connection fo	r C2
time:	
jitter_min: 600	
jitter_max: 1200	
options:	
<pre>dest_ip: \$first_dns_response.0</pre>	
proto: tcp	
dest_port: 443	
service: http,ssl	
spawns:	
– event: salo.events.zeek.SSLModel	
time:	
jitter_min: 0	
jitter_max: 1	
options:	
server_name: \$first_dns_query	

scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
src_ip: 192.168.54.36
time:
jitter_min: 0
jitter_max: 1
cadence: "* 10 1 * * *"
sessions:
– event: salo.events.zeek.ConnModel
description: Generate initial DNS connection and query
repeat: 1
options:
dest_port: 53
proto: udp
service: dns
spawns:
– event: salo.events.zeek.DNSModel
likelihood: 100
save_values:
first_dns_response: dns_rdata
first_dns_query: dns_query
query_timestamp: timestamp
options:
dest_port: 53
dns_rcode: 0
dns_rcode_name: NOERROR
dns_qtype: 1
dns_qtype_name: A
spawns:
- event: salo.events.suricata.dns.DNSModel
time:
options:
timestamp: \$query_timestamp
dns_type: answer

Spawned Events

– event: salo.events.zeek.ConnModel	
description: Initiate initial TLS connection for	C2
time:	
jitter_min: 600	
jitter_max: 1200	
options:	
<pre>dest_ip: \$first_dns_response.0</pre>	
proto: tcp	
dest_port: 443	
service: http,ssl	
spawns:	
– event: salo.events.zeek.SSLModel	
time:	
jitter_min: 0	
jitter_max: 1	
options:	
server_name: \$first_dns_query	

scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
src_ip: 192.168.54.36
time:
jitter_min: 0
jitter_max: 1
cadence: "* 10 1 * * *"
sessions:
– event: salo.events.zeek.ConnModel
description: Generate initial DNS connection and query
repeat: 1
options:
dest_port: 53
proto: udp
service: dns
spawns:
- event: salo.events.zeek.DNSModel
likelihood: 100
save_values:
first_dns_response: dns_rdata
first_dns_query: dns_query
query_timestamp: timestamp
options:
dest_port: 53
dns_rcode: 0
dns_rcode_name: NOERROR
dns_qtype: 1
dns_qtype_name: A
spawns:
- event: salo.events.suricata.dns.DNSModel
time:
options:
timestamp: \$query_timestamp
dns_type: answer

Saved Values

event: salo.events.zeek.ConnModel
description: Initiate initial TLS connection for C2
time:
jitter_min: 600
jitter_max: 1200
options:
dest_ip: \$first_dns_response.0
proto: tcp
dest_port: 443
service: http,ssl
spawns:
- event: salo.events.zeek.SSLModel
time:
jitter_min: 0
jitter_max: 1
options:
server_name: \$first_dns_query

cenario: Random Beacon	
uthor: Marcus LaFerrera	
late: 2021-09-25	
lescription: Generate a random host resolving a domain and then beaconing over	TLS
ptions:	
src_ip: 192.168.54.36	
ime:	
jitter_min: 0	
jitter_max: 1	
# cadence: "* 10 1 * * *"	
essions:	
– event: salo.events.zeek.ConnModel	
description: Generate initial DNS connection and query	
repeat: 1	
options:	
dest_port: 53	
proto: udp	
service: dns	
spawns:	
- event: salo.events.zeek.DNSModel	
likelihood: 100	
save_values:	
first_dns_response: dns_rdata	
first_dns_query: dns_query	
query_timestamp: timestamp	
options:	
dest_port: 53	
dns_rcode: 0	
dns_rcode_name: NOERROR	
dns_qtype: 1	
dns_qtype_name: A	
spawns:	
- event: salo.events.suricata.dns.DNSModel	
time:	
options:	
timestamp: \$query_timestamp	
dns_type: answer	

Recursive Spawning

event: salo.events.zeek.ConnModel		
description: Initiate initial TLS connection for C2		
time:		
jitter_min: 600		
jitter_max: 1200		
options:		
<pre>dest_ip: \$first_dns_response.0</pre>		
proto: tcp		
dest_port: 443		
service: http,ssl		
spawns:		
- event: salo.events.zeek.SSLModel		
time:		
jitter_min: 0		
jitter_max: 1		
options:		
<pre>server_name: \$first_dns_query</pre>		

scenario: Random Beacon
author: Marcus LaFerrera
date: 2021-09-25
description: Generate a random host resolving a domain and then beaconing over TLS
options:
src_ip: 192.168.54.36
time:
jitter_min: 0
jitter_max: 1
cadence: "* 10 1 * * *"
sessions:
– event: salo.events.zeek.ConnModel
description: Generate initial DNS connection and query
repeat: 1
options:
dest_port: 53
proto: udp
service: dns
spawns:
- event: salo.events.zeek.DNSModel
likelihood: 100
save_values:
first_dns_response: dns_rdata
first_dns_query: dns_query
query_timestamp: timestamp
options:
dest_port: 53
dns_rcode: 0
dns_rcode_name: NOERROR
dns_qtype: 1
dns_qtype_name: A
spawns:
- event: salo.events.suricata.dns.DNSModel
time:
options:
timestamp: \$query_timestamp
dns_type: answer

Multiple Events in Session

- event: salo.events.zeek.ConnModel	
description: Initiate initial TLS connection for C2	
time:	
jitter_min: 600	
jitter_max: 1200	
options:	
<pre>dest_ip: \$first_dns_response.0</pre>	
proto: tcp	
dest_port: 443	
service: http,ssl	
spawns:	
– event: salo.events.zeek.SSLModel	
time:	
jitter_min: 0	
jitter_max: 1	
options:	
server_name: \$first_dns_query	

Complex data scaffolding

class SunBurstDNSQuery(SaloStencilModel):

sunburst_phase: Literal["kill", "beacon", "preactivation"] = Field(default="beacon")
dns_query: Optional[str]
dest_port: int = Field(default=53)
dns_rcode: int = Field(default=3)
dns_rcode: int = Field(default=0)
dns_rcode_name: str = Field(default="NOERROR")
dns_qtype: int = Field(default=1)
dns_qtype.name: str = Field(default="A")
dns_qtype.name: str = Field(default="A")
dns_rdata: Optional[Union[str, List[str]]] = None
proto: str = Field(default="udp")
service: str = Field(default="dns")

@validator("dns_query", pre=True, always=True)
def set_dns_query(cls, v):
 if not v:
 region = random.choice(REGIONS)
 encoded_str = base64.b32encode(
 fake.pystr(min_chars=15, max_chars=15).encode()

domain = (

f"{encoded_str.decode().lower()}.appsync-api.{region}.avsvmcloud.com"

return domain

return v

@validator("dns_rdata", pre=True, always=True)
def set_dns_rdata(cls, v, *, values):
 if not v:
 phase = values.get("sunburst_phase")
 netblock = IPv4Network(random.choice(PHASES[phase]))
 ip = str(netblock[random.randint(0, netblock.num_addresses)])
 return ip
 return v



stencil

recipe

<pre>ss SubsritUNSQuery(SalotienclUNsQuery)s interview manufactures subsritUnsquery (http://www.metrasc.uterrorg/020/12/wasiwe-itaker-leverage-solaruinde-usply-chain-compromise-vie dis_prote in = Field(defaults) dis_crobe: int = Field(defaults) dister: int = Field(defaults) dister:</pre>		comprise SunthatioNinds
<pre>subtort_base: Literil["kill", "beacon", "preactivation"] = Field(default="beacon") dsts_uper; @hitesinelises. dsts_int = Field(default=3) dsts_int = Field(default=4) dsts_int = Field(default=4)</pre>	ass_SunBurstDNSQuery(SaloStencilModel):	author Marciel Saprara
<pre>ds.gury: 0ptional[str] dss.pury: 0ptional[str] ds</pre>	<pre>sunburst_phase: Literal["kill", "beacon", "preactivation"] = Field(default="beacon")</pre>	data: 201-11-16
<pre>dst_orf: int = field(default=3) dns_rcode_mame: str = field(default=4) dns_rcode_mame: str = field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_field(default=4) dns_rcode_f</pre>	dns query: Optional[str]	
<pre>dms_rcdc: int = field(default=%) dms_rcdc: got_max: str = field(default=%DERMOR*) dms_rdcto: funt = field(default=%DERMOR*) excluse : str = f</pre>	dest port; int = Field(default=53)	 - https://www.fireeve.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-wit
<pre>ds_rccd_max: str = Field(default='NOEROR") ds_rctype: int = Field(default='A') def set_ds_query(is, v): if not v: return int = random.choice(REGIOUS) encoded_str = base64.b32encode(f ske.pystr(sin_chars=15).encode()) domain = (f"(encoded_str.decode().lower()).appsync-api.(region).avswncloud.com' f"(encoded_str.decode().lower()).appsync-api.(region).avswncloud.com' f"(encoded_str.decode().lower()).appsync-api.(region).avswncloud.com' f"(encoded_str.decode().lower()).appsync-api.(region).avswncloud.com' f" f= striction(rint=', region).avswncloud.com' f"(encoded_str.decode().lower()).appsync-api.(region).avswncloud.com' f" f= striction(rint=', region).avswncloud.com' f" f= stricti</pre>	dns rcode: int = Field(default=0)	– https://www.netresec.com/?page=Blog&month=2020-12&post=Reassembling-Victim-Domain-Fragments-from-SUNBURST-DNS
<pre>undindinty: int = Field(default=1) dns_dtype_inne: str = Field(default=4**) dns_dtype_inne: str = Field(default=4**) dns_dtype_inne: str = Field(default=4**) dns_dtype_inne: str = Field(default=4**) groto: str = Field(default=4**) groto: str = Field(default=4**) gevidator(*dns_query(is, v): if not v: region = random.choice(KEGOMS) encoded_str = base64.b32encode(field(state the response IP address as outlined in the netreese blog post in references suburs(IMSGomery Stencil to populate the appropriate fields in the spanned events freturn v gevidator(*dns_rdata(is, v, *, values): if not v: region = andom.choice(FMASES[phase])) if net v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return v gevidator(*dns_rdata(is, v, *, values): if not v: return ip return v gevidator(*dns_rdata(is, v, *, values):) if not v: return v return v return v return v return v return v return ip return v return v return ip re</pre>	dns_rode name: str = Field(default="NOERBOR")	description: Solarwinds host resolving domain and beaconing over TLS
<pre>uns_type: int = fright(default="A") dns_tdis_dista: Optional[Union[str, List[str]]] = None proto: str = Field(default="A") def set_dns_query(", pre=True, always=True) def set_dns_query(cls, v): if not v: region = random.choice(REGIONS) encoded_str = base64.b32encode(f Ake.pystr(mi_chars=15, max_chars=15).encode() j domain = (f"(encoded_str.decode(), lower()).appsync-api.(region).avswacloud.com" j return domain return domain return v</pre>	dns_reduc_ndme. set = Field(default=1)	options:
<pre>uns_dype_late: St = preducted atter = y = definite: St = definit: St = definit: St = defi</pre>	des etuna ennet eta = Eiald/default=UAU)	src_ip: 192.168.54.36
<pre>time: if not v: file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefault="idefaulte") file(idefaulte") file(idefaulte")</pre>	dns_qtype_name: Str = Field(deradit= A)	dest_ip: 167.114.213.199
<pre>start: "2020-12-24T0001153.000004" start: "2020-12-24T0001153.00004" start: "2020-12-24T0001</pre>	dns_rdata: Optionat[Union[str, List[str]]] = None	time:
<pre>service: str = Field(default="dns") gvalidator("dns_query", pre=True, always=True) def set_dns_query(ls, v): if not v: region = random.choice(REGIONS) encoded_str = base64.b32encode(fake.pystr(min_chars=15, max_chars=15).encode()) domain = (f"(encode_str.decode().lower()).appsync-api.{(region).avswncloud.com") return domain return domain return v def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netbleck=TFVAktork(random.choice(PMASES[phase])) ip = str(netbleck[rundom.randint(0, netblock.num_addresses])) return ip return v v return ip return ip return ip return ip return v v return ip return v return ip return i</pre>	proto: str = Field(default="udp")	start: "2020-12-24T00:01:59.000004"
<pre>gvalidator("dns_query", pre=True, always=True) def set_dns_query(Cis, v): if not v: region = random.choice(REGIONS) encoded_str = base64.b32Recode(fake.pystr(min_chars=15, max_chars=15).encode() e</pre>	service: str = Field(default="dns")	jitter_min: 0
<pre>gvalidator("dns_query", pre=True, always=True) def set_dns_query(cls, v): if not v: region = random.choice(REGTONS) encoded_str = base64.b32encode(fake.pystr(min_chars=15).encode() / domain = (f"(encoded_str.decode().lower()).appsync-api.(region).avsymcloud.com") return domain return v evalidator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Metwork(random.choice(PMASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses])) return v return v return v return v return v return v return v return v return ip return v return v return v return v return v return sip return v return sip return</pre>		jitter_max: 1
<pre>def set_dns_query(cls, v): if not v: region = random.choice(REGIONS) encoded_str = base64.b32encode(fake.pystr(min_chars=15, max_chars=15).encode()) domain = (f"(encoded_str.decode().lower()}.appsync-api.{region}.avsymcloud.com") domain = (f"(encoded_str.decode().lower()}.appsync-api.{region}.avsymcloud.com") return domain return domain return v (walidator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPvANetwork(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v return v def set_nry v def set_nry v def set_nry v def set_nry v def set_nry.timestamp: squery_timestamp if not v: phase = values.get("sunburst_phase") netblock = IPvANetwork(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v return v return v return v return v def set_nry.timestamp: squery_timestamp spanns: def set_nry.timestamp: squery_timestamp spanns: def set_nry.timestamp; squery_timestamp spanns: def set_nry.timestamp; squery_timestamp spanns: def set_nry,timestamp; squery_timestamp spanns: def set_nry,timestamp; squery_timestamp spanns; def set_nry,timestamp; squery_timestamp spanns; def set_nry,timestamp squery_timestamp squery_timestamp squery_timestamp def squery timestamp squery_timestamp squery_timestamp def squery timestamp def squery timestamp def squery timestamp def squery timestamp def squery timestamp return v def sque</pre>	@validator("dns_query", pre=True, always=True)	# cadence: "* 10 1 * * *"
<pre>if not v: region = random.choice(REGIONS) encoded_str = base64.b32encode(fake.pystr(min_chars=15, max_chars=15).encode()) domain = (f"(encoded_str.decode().lower()).appsync-api.(region).avsvmcloud.com") return domain return v event: salo.seconts.zeek.comModel def set_dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): inetblock = IPv4Network(random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): inetblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): inetblock[random.randint(0, netblock.num_addresses)]) return v event v def set_dns_rdata(cls, v, *, values): inetblock[random.randint(0, netblock.num_addresses)]) return v event salo.seents.zeek.DNSModel time: options: timestamp spawser def set_dns_rdata(cls, v, *, values): inetblock[random.randint(0, netblock.num_addresses)]) return v def set_dns_rdata(cls, v, *, values): inetblock[random.randint(0, netblock.num_addresses)]) return v def set_dns_rdata(cls, v, *, values):</pre>	<pre>def set_dns_query(cls, v):</pre>	sessions:
<pre>region = random.choice(REGIONS) encoded_str = base64.b32encode(fake.pystr(min_chars=15, max_chars=15).encode() domain = (f"(encoded_str.decode().lower()).appsync-api.(region).avsymcloud.com") return domain return v @validator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PMASES[phase])) ip = str(netblock(random.randint(0, netblock.num_addresses)]) return v </pre>	if not v:	* we will teverage the sumbursionsquery stellar to randomizy generate box queries that are similar to those that sumbursi # generated (i) a perfectives processes and the approvement of a perfective of a perfective of a perfective of the sum of a perfective of the
<pre>encoded_str = base64.b32encode(fake.pystr(min_chars=15, max_chars=15).encode()) domain = (f"{encoded_str.decode().lower()}.appsync-api.{region}.avsymcloud.com") return domain return v evaluator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v ereturn v def set_nor_rdata(0, return ip return v ereturn v ereturn v ereturn v ereturn v def set_dns_rdata(0, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.randint(0, netblock.num_addresses)]) return v ereturn ereturn ereturn ereturn ereturn</pre>	<pre>region = random.choice(REGIONS)</pre>	w generation (if etc.) in Figure Divergenine Count opposition - avent: salo sterrile s
<pre>fake.pystr(min_chars=15, max_chars=15).encode() } fake.pystr(min_chars=15, max_chars=15).encode() } fake.pystr(min_chars=15).encode() } fake.pystr(min_chars=15).encode()</pre>	encoded_str = base64.b32encode(options:
<pre> / / / / / / / / / / / / / / / //</pre>	<pre>fake.pystr(min_chars=15, max_chars=15).encode()</pre>	# Possible options: kill, beacon, preactivation
<pre>domain = (f"{encoded_str.decode().lower()}.appsync-api.{region}.avsvmcloud.com" f"{encoded_str.decode().lower()}.appsync-api.{region}.avsvmcloud.com" f" {encoded_str.decode().lower()}.appsync-api.{region}.avsvmcloud.com" f" {use the data generated from the SunburstDNSQuery stencil to populate the appropriate fields in the spawned events spawns: return domain return v @validator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PHASES[phase])) return ip return ip return v def set_f("sunburst_phase")) return ip return iv return iv def set_f("sunburst_phase")) return ip return iv return iv def set_f("sunburst_phase")) return ip return v def set_f("sunburst_phase")) return ip return iv def set_f("sunburst_phase")) return iv def set_f("sunburst_phase")) return iv def set_f("sunburst_phase")) return ip return iv return iv</pre>)	# This value will dictate the response IP address as outlined in the netresec blog post in references
<pre>f"{encoded_str.decode().lower()}.appsync-api.{region}.avsvmcloud.com"</pre>	domain = (sunburst_phase: beacon
<pre>spawns: return domain return v @validator("dns_rdata", pre=True, always=True) def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return ip return v </pre>	f"{encoded str.decode().lower()}.appsync-api.{region}.avsymcloud.com"	# Use the data generated from the SunburstDNSQuery stencil to populate the appropriate fields in the spawned events
<pre>return domain return domain return v @validator("dns_rdata", pre=True, always=True) @validator("dns_rdata", pre=True, always=True) @def set_dns_rdata(cls, v, *, values): if not v: phase = values.get("sunburst_phase") netblock = IPv4Network(random.choice(PHASES[phase])) ip = str(netblock[random.randint(0, netblock.num_addresses)]) ip = str(netblock[random.randint(0, netblock.num_addresses)]) return v</pre>)	spawns:
return v description: Generate initial DNS connection and query return v spawns: @validator("dns_rdata", pre=True, always=True) - event: salo.events.zeek.DNSModel geveltidator("dns_rdata", pre=True, always=True) save_values: def set_dns_rdata(cls, v, *, values): query_timestamp: timestamp if not v: spawns: phase = values.get("sunburst_phase") # Spawn the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip options: return v dns_type: answer	return domain	- event: salo.events.zeek.ConnModel
return v spawns: @validator("dns_rdata", pre=True, always=True) - event: salo.events.zeek.DNSModel def set_dns_rdata(cls, v, *, values): avery_timestamp: timestamp if not v: guery_timestamp: timestamp phase = values.get("sunburst_phase") # Spawn the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) # Spawn the suricata DNSModel ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip time: return v dns_type: answer		description: Generate initial DNS connection and query
@validator("dns_rdata", pre=True, always=True) - event: salo.events.zeek.DNSModel @def set_dns_rdata(cls, v, *, values): save_values: if not v: query_timestamp: timestamp phase = values.get("sunburst_phase") * Spawn the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses]]) options: return ip timestamp: \$query_timestamp return v dns_type: answer		spawns:
def set_dns_rdata", pre=rrue, always=rrue; save_values: def set_dns_rdata(cls, v, *, values): guery_timestamp: timestamp if not v: * Spawn the suricata DNSModel so we can ensure consistency across generated events phase = values.get("sunburst_phase") # Spawn the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip timestamp: \$query_timestamp return v dns_type: answer	Qualidates (Ildas adatalli ana Taus al austrus)	- event: salo.events.zeek.unsmodel
def set_dns_rdata(cls, v, *, values): if not v: spawns: if not v: spawns: phase = values.get("sunburst_phase") # Spawn the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip timestamp: \$query_timestamp return v dns_type: answer	(validator("dns_rdata", pre=irue, always=irue)	save_values:
if not v: phase phase = values.get("sunburst_phase") # Span the suricata DNSModel so we can ensure consistency across generated events netblock = IPv4Network(random.choice(PHASES[phase])) + event: salo.events.suricata.dns.DNSModel ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip timestamp: \$query_timestamp return v dns_type: answer	der set_dns_rdata(cls, v, *, values):	query_linestamp. (inestamp snawns:
phase = values.get("sunburst_phase") - event: salo.events.suricata.dns.DNSModel netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip timestamp: \$query_timestamp return v dns_type: answer	if not v:	apomis. # Spawn the surjects DNSModel so we can ensure consistency across generated events
netblock = IPv4Network(random.choice(PHASES[phase])) time: ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip timestamp: \$query_timestamp return v dns_type: answer	<pre>phase = values.get("sunburst_phase")</pre>	 event: salo.events.suricata.dns.DNSModel
<pre>ip = str(netblock[random.randint(0, netblock.num_addresses)]) options: return ip return v dns_type: answer</pre>	<pre>netblock = IPv4Network(random.choice(PHASES[phase]))</pre>	time:
return ip timestamp: \$query_timestamp return v dns_type: answer	<pre>ip = str(netblock[random.randint(0, netblock.num_addresses)])</pre>	options:
return v dns_type: answer	return ip	timestamp: \$query_timestamp
	return v	dns_type: answer

class SunBurstDNSQuery(SaloStencilModel)

sunburst_phase: Literal["kill", "beacon", "preacti dns_query: Optional[str] dest_port: int = Field(default=53) dns_rcode: int = Field(default=0) dns_rcode_name: str = Field(default="NOERROR") dns_qtype: int = Field(default=1) dns_qtype_name: str = Field(default="A") dns_rdata: Optional[Union[str, List[str]]] = None proto: str = Field(default="udp") service: str = Field(default="udp")

domain = (
 f"{encoded_str.decode().lower()}.apps

return domain

return v

validator("dns_rdata", pre=True, always=True)
ef set_dns_rdata(cls, v, *, values):
 if not v:
 phase = values.get("sunburst_phase")
 netblock = IPv4Network(random.choice(PHASE
 ip = str(netblock[random.randint(0, netblo
 return ip
 return ip

"id.orig_p": 58327, "id.resp h": "167.114.213.199", "id.resp_p": 53, "trans_id": 33083, "rtt": 0.90494320944503. "query": "pjugi4dsljcuezkmobdg6wcq.appsync-api.us-east-2.avsvmcloud.com", "gclass": 1, "gclass name": "C INTERNET", "rcode": 0, "rcode name": "NOERROR", "AA": true. "TC": false, "RD": false, "RA": false. "Z": 0, "answers": ["87.238.86.32" 19518 "rejected": false "timestamp": "2020-12-24T00:02:03.974343", "src_port": 58327, "dest_ip": "167.114.213.199", "dest port": 53, "dns": { "id": 33083, "ar": true.

"duration": 8.71176888606152,

"orig_bytes": 7368,
"resp_bytes": 2558,
"conn_state": "S0",
"missed_bytes": 3978,
"history": "S",
"orig_pkts": 1334,
"orig_ip_bytes": 3571,
"resp_pkts": 9453,
"resp_pkts": 2897

"aa": true,
"tc": false,
"rd": false,

sive-attacker-leverages-solarwinds-supply-chain-compromises-w wassembling-Victim-Domain-Fragments-from-SUNBURST-DNS over TLS

ly generate DNS queries that are similar to those that SUNBURS" is-east-1.avsvmcloud.com)

outlined in the netresec blog post in references

l to populate the appropriate fields in the spawned events

consistency across generated events



Individual log objects

class ConnModel(ZeekModel):

_refs: List[str] = [
 "https://docs.zeek.org/en/master/scripts/base/protocols/co
 "https://docs.zeek.org/en/master/logs/conn.html",

proto: str = Field(...) service: str = Field(...) duration: Optional[float] = None orig_bytes: int = Field(default_factory=fake.pyint) resp_bytes: int = Field(default_factory=fake.pyint) conn_state: Optional[str] = None local_orig: Optional[bool] = None local_resp: Optional[bool] = None missed_bytes: int = Field(default_factory=fake.pyint) history: Optional[str] = None orig_pkts: int = Field(default_factory=fake.pyint) orig_ip_bytes: int = Field(default_factory=fake.pyint) resp_pkts: int = Field(default_factory=fake.pyint) resp_ip_bytes: int = Field(default_factory=fake.pyint) tunnel parents: Optional[str] = None orig_l2_addr: Optional[str] = None resp_l2_addr: Optional[str] = None vlan: Optional[int] = None inner_vlan: Optional[int] = None speculative_service: Optional[str] = None

Built with pydantic*

models define the structure of log events

Native Model Features

- Data validation
- Type Enforcement
- Field aliases
- Randomized field content

ss DNSModel(ZeekModel):	
<pre>_refs: List[str] = [</pre>	
"https://docs.zeek.o	rg/en/master/scripts/base/protocols/dns/main.zeek.html#type-DNS::Info"
"https://docs.zeek.o	rg/en/master/logs/dns.html",
proto: Optional[str] = No	one
<pre>dns_id: Optional[int] = !</pre>	lone
<pre>dns_rtt: Optional[float]</pre>	= None
dns_query: str = Field(de	efault_factory=fake.hostname)
dns_qclass: int	
dns_qclass_name: str	
dns_qtype: int	
dns_qtype_name: str	
dns_rcode: int	
dns_rcode_name: str	

class Config:

```
json_encoders = {datetime: lambda v: f"{v.isoformat()}Z"}
fields = {
    "timestamp": "ts",
    "src_ip": "id.orig_h",
    "src_port": "id.orig_p",
    "dest_ip": "id.resp_h",
    "dest_port": "id.resp_p",
}
```

```
class ConnModel(ZeekModel):
    refs: List[str] = [
        "https://docs.zeek.org/en/master/scripts/base/protocols/co
        "https://docs.zeek.org/en/master/logs/conn.html",
    proto: str = Field(...)
    service: str = Field(...)
    duration: Optional[float] = None
    orig_bytes: int = Field(default_factory=fake.pyint)
    resp_bytes: int = Field(default_factory=fake.pyint)
    conn_state: Optional[str] = None
    local_orig: Optional[bool] = None
    local_resp: Optional[bool] = None
    missed_bytes: int = Field(default_factory=fake.pyint)
    history: Optional[str] = None
    orig_pkts: int = Field(default_factory=fake.pyint)
    orig_ip_bytes: int = Field(default_factory=fake.pyint)
    resp_pkts: int = Field(default_factory=fake.pyint)
    resp ip bytes: int = Field(default factory=fake.pyint)
    tunnel parents: Optional[str] = None
    orig_l2_addr: Optional[str] = None
    resp_l2_addr: Optional[str] = None
    vlan: Optional[int] = None
    inner vlan: Optional[int] = None
```

```
speculative_service: Optional[str] = None
```



Event Models Available

Zeek

- conn
- dns
- files
- http
- rdp
- smtp
- ssl

Suricata

- dns
- http

Sysmon

• eventcode 3

GitHub Audit

- business
- git
- hook
- integrations
- org
- repo
- repository
- team



console | log file | Splunk

Suricata # salo.events.suricata: outputs: file: path: suricata/eve.log splunk: index: salo sourcetype: suricata # # Zeek # salo.events.zeek.conn.ConnModel: outputs: file: path: zeek/conn.log splunk: index: salo sourcetype: "bro:conn:json"

Flexible Output Schema

• Any log format

- JSON is natively supported
- Simple to add additional output schemas

def generate(self, by_alias: bool = True, exclude_none: bool = True):
 return self.json(by_alias=by_alias, exclude_none=exclude_none)

```
def generate(self, by_alias: bool = True, exclude_none: bool = True):
    filename = getframeinfo(currentframe()).filename
    parent = Path(filename).resolve().parent
    template = parent.joinpath(self._template)
    env = Environment(
        loader=FileSystemLoader(parent),
        trim_blocks=True,
        lstrip_blocks=True,
        autoescape=select_autoescape(default_for_string=True, default=True),
    )
    return env.get_template(template.name).render(
        self.dict(by_alias=by_alias, exclude_none=exclude_none)
```

Roadmap Wishlist

- Community involvement
- More recipes and stencils
- Additional event models

SALO https://github.com/splunk/salo

Questions?

Marcus LaFerrera | @mlaferrera | mlaferrera@splunk.com

SALO

code

https://github.com/splunk/salo

documentation https://splunk.github.io/salo