DON’T GET HACKED, GET AMINER
Log Data Analysis for Intrusion Detection

Florian Skopik, Markus Wurzenberger, Max Landauer

In-Depth Security Conference Europe (DeepSec), 19.11.2021
WHAT DO THEY HAVE IN COMMON?
MONITORING TODAY

• Complex systems organically grow bottom up
  • Implementation and configuration failures lead to vulnerabilities
  • Design errors cause weaknesses
  • Prevention ultimately fails
• Monitoring focuses on the early discovery of adversarial actions, such as the exploitation of vulnerabilities and weaknesses
• Today’s state of the art still is:
  • Mainly investigation of network traffic (common tools)
  • Signature-based search for known bad
  • Log data investigation with SIEMs – mostly predefined rules only
  • Limited anomaly detection (mostly outliers, e.g. uncommon protocols)
ISSUES WITH NETWORK SECURITY MONITORING

- Increasing use of encryption drastically limits the visibility on the network
  - TLS everywhere
  - Even for DNS: DoH, DoT on the horizon
  - Focus on limited initial handshake (TLS 1.3), netflows
- Attackers “living off the land“
  - Use of built-in tools (cmd.exe, powershell.exe etc.)
  - Use of standard protocols
- High-profile attacks often do not exploit any technical vulnerabilities at all
  - Social engineering
  - Compromise of legitimate update mechanisms, root certificates etc.
  - Hence, no signature can capture that behavior, no rule can flag malicious activities
A WAY OUT: ANOMALY DETECTION

• However, **attackers utilize systems differently** from legitimate users…
  • Access other DMZ servers from a compromised web server
  • Use of SSH maintenance interface instead of the web interface
  • Login using backup system SSH key intended for SFTP file transfer
• Novel **machine learning** approaches
  • **Observe a system** and its “normal” utilization
  • Dynamically **build up** a model that constitutes a **baseline**
  • **Alert on** significant **deviations** from this baseline
• **Visibility** of adversarial actions is key!
  • On the **endpoint**
  • Use what we have – no additional agent
  • **Verbose log data** from services, application, operating systems

Know thyself.
- Socrates

22/11/2021
### INTRUSION DETECTION: TECHNIQUES OVERVIEW

<table>
<thead>
<tr>
<th>Technique</th>
<th>Blocked Addresses</th>
<th>Monitored Logs</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Signature-based detection  | 192.168.141.10, 192.168.176.23         | 10.237.2.50, 10.237.2.22, 10.237.2.50, 192.168.175.131, 192.168.176.23         | - Efficient  
- Only known bad values  
- Variants |
| Allowlisting               | 10.237.2.50, 172.28.193.48, 10.237.2.22| 10.237.2.50, 10.237.2.22, 10.237.2.50, 172.28.195.6, 172.28.193.48             | + Exact  
+ Detects unknown attacks  
- Complex to generate/maintain |
| Anomaly Detection          | 10.237.2.50, 10.237.2.22, 10.237.2.50, 172.28.194.6, 172.28.193.48              | + Detects unknown attacks  
+ Self-learning               |
- Training phase (benign behavior)  
- False positives |
MAKE LOG DATA ANALYSIS SMART!

- Log data are **textual data**, not simple numerical values
- Log data have **mostly unknown structure** and unknown meaning
- For intrusion detection, log data need to be **processed online** („single pass approaches“)
- Observed systems change frequently (updates, extensions, etc.), leading to a **moving baseline**.

**SOLUTIONS in this presentation:** Machine Learning and AI for …
- Part I: Flexible creation of log data parsers
- Part II: Online anomaly detection beyond simple outlier detection

https://www.amazon.de/dp/3030744493
COMMON LOG FORMATS

- Apache Access logs
  - Structured
  - Many Categorical variables
  - Some complex tokens (session Ids, …)
COMMON LOG FORMATS

- Audit logs
- Key-value pairs
- High granularity
- Encoded values

Access logs

22/11/2021
COMMON LOG FORMATS

- Event logs
  - Json format
  - Many attributes, high volume

Access logs

Audit logs
COMMON LOG FORMATS

- **System logs**
  - Unstructured (human readable messages)
  - Diverse event formats
  - Correlations (Starting `<service>` → Started `<service>`)

**COMMON LOG FORMATS**

- Access logs
- Audit logs
- Event logs
COMMON LOG FORMATS

- DNS logs
  - User queries

---

System logs

Access logs

Audit logs

Event logs
COMMON LOG FORMATS

- **Application logs**
  - E.g., horde mail logs
  - User behavior in specific context
COMMON LOG FORMATS

• Many formats
  • No common standards
• Diverse events
  • Creating parsers can be tough!
  • No well-defined syntax
  • No complete list of events available
• Two main goals of log parsing:
  • Differentiate events
  • Access specific tokens
INTRODUCING ÆCID AND AMINER

- **ÆCID** is a mature intrusion detection system using computer log data
  - Ingests **log data from any system**
  - Works with domain specific and previously unknown systems, i.e., does not rely on predefined parsers – **self-learning**!
  - **Light-weight**, distributed anomaly detection
  - Clients run with **low memory footprint** and minimum CPU utilization
  - **Not in competition** with well-established systems, but as **additional detection mechanism**
    - Proof-of-Concept deployments as **sensors** for **ELK Stack** and **QRadar SIEMs**
THE ÆCID APPROACH

1. **Log parser** generation
   - A “recipe” on how to dissect log lines of unknown grammar
   - Make log data usable for analysis → structured representation & easy to access

2. **Hypotheses** proposal
   - Distribution of property values (e.g., IP addresses, user names, …) in single events
   - And across multiple events
   - Correlation of event types

3. **Rule** generation through continuous hypotheses evaluation
   - Sort out unstable hypotheses and create rules for stable ones
   - Constitution of the system behavior model (learned behavior model)

4. **Anomaly detection**: rate the deviation of actual system behavior from the learned behavior model (anomalous points / context / frequency / sequence of events)
PART I:
FLEXIBLE CREATION OF LOG DATA PARSERS
FAST LOG DATA PARSING

- Using regex for parsing is inefficient → O(n)
- Represent log line model as tree-like graph → O(log(n)) → Parse data once!
  - Describes information most efficiently – with minimal storage requirements
  - Efficient log line processing, classification and information access

How to generate an efficient tree-like log parser?

Oct 15 00:10:27 mymachine ntpd[16721]: Listen normally on 2 lo 127.0.0.1 UDP 123
/model/syslog/time: 2021-10-15 00:10:27
/model/syslog/host: mymachine
/model/services/ntpd/pid: ntpd
/model/services/ntpd/listen/fd: 2
/model/services/ntpd/listen/if: 127.0.0.1
AECID-PG – CONCEPT

• Density based approach
• Independent from semantics
• Detect static and variable log line parts
• Build model from a local point of view
  • Different log line classes should not influence each other
• Prohibit overfitting
AECID-PG: PARSER GENERATOR

- Nodes:
  - Static
  - Variable
  - Optional
  - Deleted

```
DATETIME A E G
DATETIME A E H J
DATETIME B E I K
DATETIME B E I
DATETIME C F 1
DATETIME C F 2
DATETIME D 1 I
DATETIME D 2 I
DATETIME L M N O
```
SAMPLE LOGS

ntpd [16721]: Listen and drop on 0 v4wildcard 0.0.0.0 UDP 123
ntpd [16721]: Listen and drop on 1 v6wildcard :: UDP 123
ntpd [16721]: Listen normally on 2 lo 127.0.0.1 UDP 123
ntpd [16721]: Listen normally on 3 eth0 134.74.77.21 UDP 123
ntpd [16721]: Listen normally on 4 eth1 10.10.0.57 UDP 123
ntpd [16721]: Listen normally on 5 eth1 fe80::5652:ff:fe5a:f89f UDP 123
ntpd [16721]: Listen normally on 6 eth0 fe80::5652:ff:fe01:1ff UDP 123
ntpd [16721]: Listening on routing socket on fd #24 for interface updates
ntpd [16721]: Listen and drop on 0 v4wildcard 0.0.0.0 UDP 123
ntpd [16721]: Listen and drop on 1 v6wildcard :: UDP 123
ntpd [16721]: Listen normally on 2 lo 127.0.0.1 UDP 123
ntpd [16721]: Listen normally on 3 eth0 134.74.77.21 UDP 123
ntpd [16721]: Listen normally on 4 eth1 10.10.0.57 UDP 123
ntpd [16721]: Listen normally on 5 eth1 fe80::5652:ff:fe5a:f89f UDP 123
ntpd [16721]: Listen normally on 6 eth0 fe80::5652:ff:fe01:1ff UDP 123
ntpd [16721]: Listening on routing socket on fd #24 for interface updates

ntpd[§]: Listen and drop on § § § UDP 123
ntpd[§]: Listen normally on § § § UDP 123
ntpd[§]: Listening on routing socket on fd #§ for interface updates
ntpd [16721]: Listen and drop on 0 v4wildcard 0.0.0.0 UDP 123
ntpd [16721]: Listen and drop on 1 v6wildcard :: UDP 123
ntpd [16721]: Listen normally on 2 lo 127.0.0.1 UDP 123
ntpd [16721]: Listen normally on 3 eth0 134.74.77.21 UDP 123
ntpd [16721]: Listen normally on 4 eth1 10.10.0.57 UDP 123
ntpd [16721]: Listen normally on 5 eth1 fe80::5652:ff:fe5a:f89f UDP 123
ntpd [16721]: Listen normally on 6 eth0 fe80::5652:ff:fe01:1fff UDP 123
ntpd [16721]: Listening on routing socket on fd #24 for interface updates

ntpd[<int>]: Listen and drop on <int> <str> <ip> UDP 123
ntpd[<int>]: Listen normally on <int> <str> <ip> UDP 123
ntpd[<int>]: Listening on routing socket on fd #<int> for interface updates
"ntpd" : "Listening on routing socket on fd #"" int " for interface updates"

"Listen and drop on" <int> " " <str> " " <ip> " UDP 123"

"Listen normally on" <int> " " <str> " " <ip> " UDP 123"
Listen and drop on

ntpd[

Listen normally on

Listening on routing socket on fd #
"Listen and drop on" <int> " " <str> " " <ip> " UDP 123"

"ntpd[" <int> "]: " "Listen normally on" <int> " " <str> " " <ip> " UDP 123"

upd s12 fd s13

"Listening on routing socket on fd #"<int> "for interface updates"
Listen and drop on

Listen normally on

Listening on routing socket on fd #
"Listen and drop on" <int> " " <str> " " <ip> " UDP 123"

"ntpd[" <int> "]: " "Listening on routing socket on fd #" <int> "for interface updates"
Listen and drop on

Listen normally on

Listening on routing socket on fd #

for interface updates

ntpd/fm/drop/ip
PARSER-GENERATOR
Live Demo
PART II:
ONLINE ANOMALY DETECTION
HYPOTHESES PROPOSAL AND ANOMALY DETECTION

Jun 20 00:59:37 localhost sshd[1008]: Accepted public key for backup from 172.29.147.33 port 54149 ssh2: RSA SHA256:9k...

/model/syslog/time: Jun 20 00:59:37
/model/syslog/host: localhost
/model/services/sshd/sname: sshd
/model/services/sshd/msg/acceptedpk/pid: 1008
/model/services/sshd/msg/acceptedpk/user: backup
/model/services/sshd/msg/acceptedpk/originip: 172.29.147.33
/model/services/sshd/msg/acceptedpk/port: 54149
/model/services/sshd/msg/acceptedpk/protocol: ssh2
/model/services/sshd/msg/acceptedpk/crypto: RSA
/model/services/sshd/msg/acceptedpk/fingerprint: SHA256:9k...

Simple Example Hypotheses:
user{backup} ~ remoteip{172.29.147.33}
user{backup} ~ fingerprint{SHA256:9k...}
user{backup} only allowed in time_hh{[00,03]}

...• Different Methods for hypothesis generation (incl. brute force)
• Coverage of events is complex to determine
• Maximize detection capabilities with minimum number of (stable) hypotheses
• Continuous learning in parallel to detection
SCENARIO: OPERATIONAL TECHNOLOGY (OT)

- **System:** Hosts report system monitoring data: Temperature, bytes, syscalls
- **Data:** 17:05:51 server-5 systemd[1644]: Bytes sent: 162613
- **Attack:** Attacker installs crypto-miner on one of the hosts
- **Consequences:** Changes in monitoring data
- **Detection:** Value Range Detector
  - Learn minimum and maximum
  - Automatically expand and raise anomalies

server-5: 149243
server-3: 122934
server-5: 155384
server-5: 158304
server-3: 112834

server-5: [149243, 158304]
server-3: [112834, 122934]
SCENARIO: REMOTE COMMAND EXEC

- **System:** Webserver with apache access logs
- **Data:** 10.35.34.9 - - [18:33:55] "GET /wp-admin/ HTTP/1.1" 302 361 "-" "Firefox/86.0"
- **Attack:** Webshell on server allows attacker to execute commands
- **Consequences:** Commands visible in accessed resources
- **Detection:** Entropy Detector
  - Learn probability distributions of character pairs, e.g., “/wp-admin/”, “/wp-content/”
  - Automatically update probabilities during training or use default table

```
GET /wp-admin/
GET /wp-content/b.gif
GET /wp-content/hi.txt
GET /wp-admin/site.html
```

```
“w”: [“p”: 1.0]
“p”: [“-”: 1.0]
“-”: [“a”: 0.5, “c”: 0.5]
“n”: [“/”: 0.33, “t”: 0.67]
```

```
GET /wp-admin/
GET /wp-content/a.gif
GET /static/evil.php
GET /wp-content/site.html
```
**SCENARIO: PROCESS HIJACKING**

- **System:** Host running audit daemon
- **Data:** type=SYSCALL msg=audit(1583016732.264:4786292): syscall=0 success=yes
- **Attack:** Attacker modifies a process as part of a privilege escalation
- **Consequences:** Process carries out operations in different order than before
- **Detection:** Sequence Detector
  - Learns sequences of fixed lengths
  - Need to untangle interleaved processes
KEY TAKEAWAYS

- **Log data analysis** is a non-trivial task
  - Many different formats and events
  - High volume data requires efficient code
  - Live monitoring and learning needs incremental algorithms

- **AECID/AMiner** allows to …
  - … manually or semi-automatically generate parsers
  - … automatically train models in a semi-supervised manner
  - … forensically analyze log data sets
  - … detect anomalies in logs as soon as they occur
LINKS

- **AECID:** [https://aecid.ait.ac.at/](https://aecid.ait.ac.at/)
- **AMiner (Github):** [https://github.com/ait-aecid/logdata-anomaly-miner](https://github.com/ait-aecid/logdata-anomaly-miner)
  - **Tutorials:** [https://github.com/ait-aecid/logdata-anomaly-miner/wiki](https://github.com/ait-aecid/logdata-anomaly-miner/wiki)
- **AMiner (Debian):** [https://packages.debian.org/sid/misc/logdata-anomaly-miner](https://packages.debian.org/sid/misc/logdata-anomaly-miner)
- **Publications + Patents:** [https://aecid.ait.ac.at/further-information/](https://aecid.ait.ac.at/further-information/)
- **Current Projects:**

---

[Guard](guard.png)  [Pandora](pandora.png)  [Decept](decept.png)
THANK YOU!
aecid@ait.ac.at