
LF
NETWORKING
LFN Developer & Testing Forum

LF NETWORKING

LFN Developer & Testing Forum

ONAP Security

Jakarta Global Requirements and Best Practices

Pawel Pawlak, F5
Amy Zwarico, AT&T
Tony Hansen, AT&T
Muddasar Ahmed, MITRE
Robert Heinemann, MITRE
Byung-Woo Jun, Ericsson

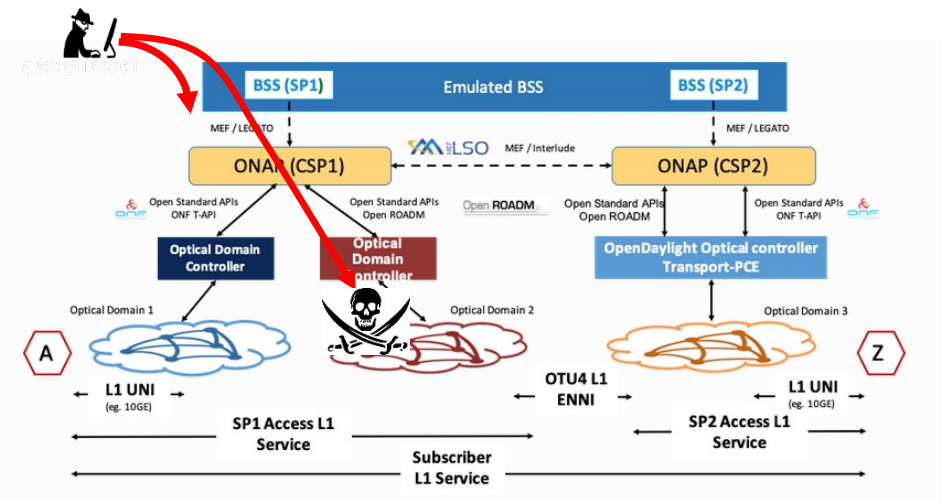
Jakarta Security Initiative & Requirements

- Initiatives
  - Standardized feature intake templates
  - Software Bill of Material (SBOM)
- Best Practices
  - Standardized fields for logging for security [REQ-1072]
  - Using basic image from Integration [REQ-1073]
- Global Requirements
  - ONAP Applications to log to STDOUT and STDERR [REQ-441 -> [REQ-1070]
  - Completion of Python Language Update (v2.7 → v3.8) [REQ-437 -> REQ-800 -> REQ-1067]
  - Completion of JAVA Language Update (v8 → v11) [REQ-438 -> REQ-801 -> REQ-1068]
  - Continuation of Packages Upgrades in Direct Dependencies [REQ-439 -> REQ-863 -> REQ-1066]
  - Continuation of Best Practices Badging* Score Improvements for Silver Level [REQ-443 -> REQ-1069]

*Please note the new naming for CII Badge is : “OpenSSF Best Practices Badge”

Logging is a Security Concern

- ONAP has a central position in the orchestration chain of the network.
- For a country or an operator, the loss of control of ONAP would have a devastating impact.
- To reduce threats, there are several **levers**: code quality, strong authentication, flow protected, etc...



ONAP MDONS Architecture example

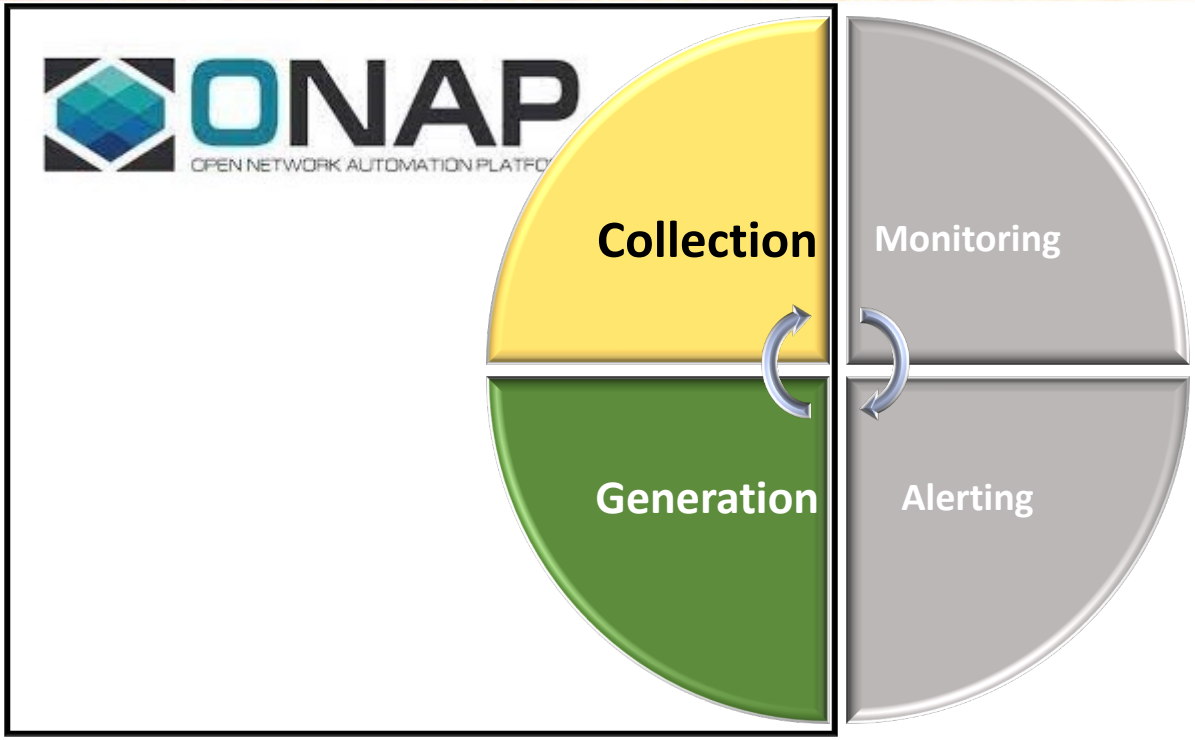
Through ONAP, the hacker could have the control to the network

Logging is one of these levers

&

Logging the right data is critical downstream to enable effective analytics

Security Logging Lifecycle



Only for ONAP Platform Components **NOT** for services orchestrated by ONAP.

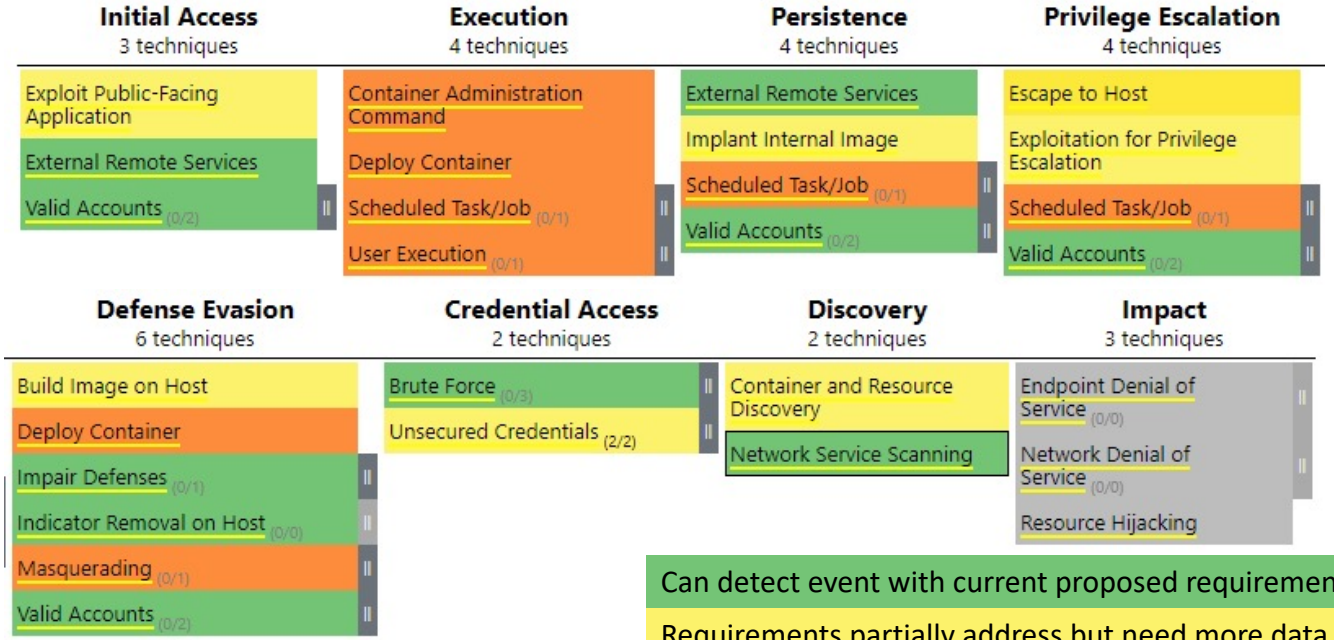
Motivation and Approach

- Operation / infrastructure teams need data to develop good security analytics.
- Challenge is to understand and anticipate what data is needed to enable those teams.
- We reviewed v9 of SECCOM container logging requirements to determine if additional requirements should be proposed to support security analytics.
- Our approach was to systemically review the v9 logging requirements against the Containers Matrix for ATT&CK® (v9) to identify gaps.

The Containers Matrix provides a list of attacker techniques that provides a convenient tool to identify data needed to craft good security analytics.

Findings: Matrix Coverage Heat Map for SECCOM Draft Logging Requirements

LF NETWORKING
LFN Developer & Testing Forum



About 36% of the Matrix is addressed by the SECCOM Container Logging Requirements

Can detect event with current proposed requirements.
Requirements partially address but need more data to detect.
Requirements do not address at all.
Outside of scope for logging.

Conclusions from Findings

- The adversary techniques listed discuss events types and log data generated from more that just the container application.
  - The pod, node (Docker) and orchestrator (K8S) are also listed.
- After systematically going through each adversary technique 5 new logging requirements were developed to address gaps.
- All proposed logging requirements are at the Docker and K8S level.
- K8S, Image Registry, and Docker daemon logs should be planned to be aggregated.
- This will allow for upwards of 85% coverage of the ATT&CK® Containers Matrix.

No new requirements at the Container Application level based on gap analysis.

Log Field Recommendations

LFN Developer & Testing Forum

15 fields total:

- 9 of 15 fields exist within the structures defined in EELF and Log Spec v1.2.
- Other 6 fields identify properties about the container itself.

Existing Fields Recommended

EELF	**BeginTimestamp OR Timestamp**	**RequestID**	**Service / Program Name**	**StatusCode**	**Category log level level**	**Severity**	**detailMessage**		
LogSpec	**LogTimestamp**	**TransactionID**					**p_message**	**p_marker**	**User**

New Fields Recommended

Container Image Name / Tag	Container Image Digest	Container ID	Container Name	Role / Attribute ID	Protocol

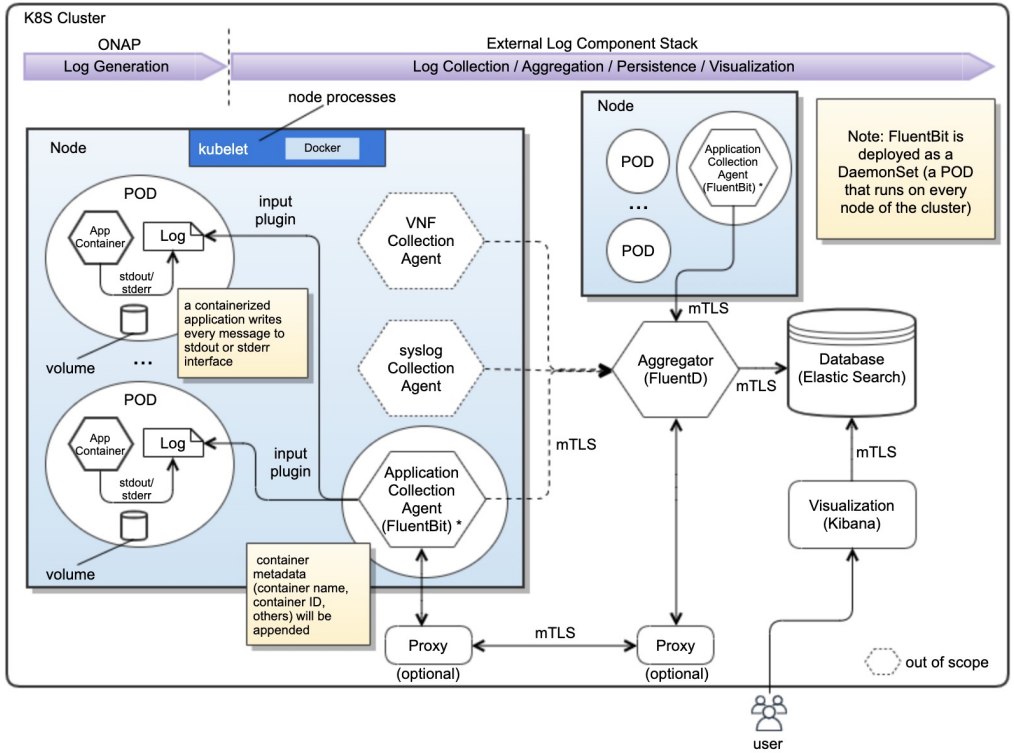
**The security basis for these log field recommendations have been derived from ONAP's VNF security requirements and MITRE's ATT&CK® Container's Matrix.*

ONAP Logging Architecture Principles

- ONAP logging architecture separates log generation from the log collection/aggregation/ persistence/visualization.
- An ONAP application should not concern itself with routing or storage of its output stream.
- Each ONAP running process writes its log data to STDOUT or STDERR.
- Archival destinations should not be visible to or configurable by the ONAP applications (separation of concerns, security reasons).
- Transferring transient local log data in the ONAP containers to the separate and centralized (or even distributed) long-term log storage is a must for security, persistent and aggregation reasons.
- ONAP supports and leverages open-source and/or standard-based logging framework for integration, extensibility and customization.
- ONAP provides logging reference implementation and allows the logging component stack is realized by choices of vendors.

ONAP Logging Architecture, Leveraging Open-Source Logging Framework

LFN Developer & Testing Forum



Note 1: all inter-component communication is secure, by leveraging service-mesh (preferred solution)

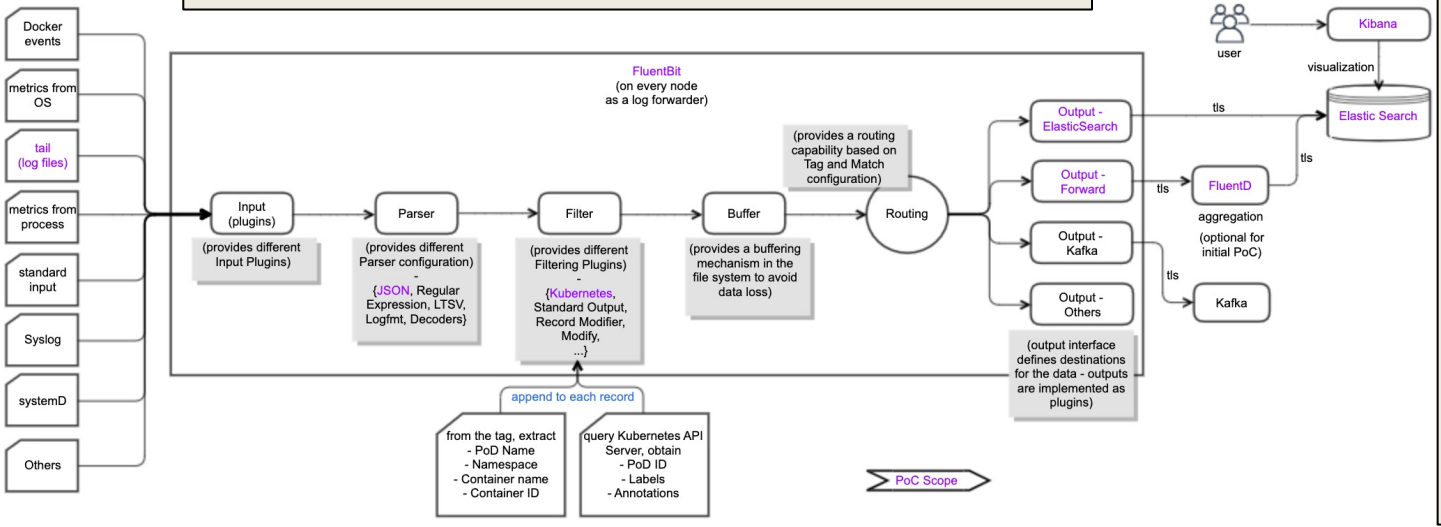
Note 2: a solution is under discussion against root access for the DaemonSet (or equivalent) configuration to make FluentBit run as non-root users

- ONAP supports open-source- and standard-based Logging architecture, separating log generation from the log collection/ aggregation/persistence/visualization.
- All ONAP components push their logs into STDOUT/STDERR, so any standard log pipe can work on the logs.
  - Allowing the logging component stack is realized by choices of vendors
  - ONAP provides a reference implementation/choice
- ONAP logs will be exported to a different and centralized location for security, persistent and aggregation reasons
  - Log collector sends logs to the aggregator in a different container
  - Aggregator sends logs to the centralized database in a different container
- Logging Functional Blocks:
  - Collector/forwarder (one per K8S node)
  - Aggregator (few per K8S cluster)
  - Database (one per K8S cluster) – could use multiple PODs for HA
  - Visualization (one per K8S cluster)
- ONAP reference implementation choice:
  - EFK: Elastic Search, FluentBit, FluentD, Kibana
- ONAP logging conforms to SECCOM Container Logging requirements
  - Standardized Logging Fields that are proposed as a best practice, plus recommended container metadata
  - https://wiki.onap.org/display/DW/Jakarta+Best+Practice+Proposal+for+Standardized+Logging+Fields

ONAP Logging Reference Implementation

LF NETWORKING
LFN Developer & Testing Forum

ONAP provides reference implementation, but the implementation can be overwritten by vendors, by leveraging their own logging stack



- When FluentBit runs, it will read, parse and filter the logs of every POD and could enrich each entry with the following information (metadata):
  - POD Name & ID
  - Container Name & ID
  - Labels & Annotations
  - Others
- To obtain this information, a FluentBit built-in filter plugin called “Kubernetes” talks to the Kubernetes API server to retrieve relevant information. All of this is handled automatically, no intervention is required from a configuration aspect.

As a log collector/forwarder, FluentBit (node-level logging agent) needs to be run on every node to collect logs from every POD; one way is FluentBit is deployed as a DaemonSet (i.e., its POD that runs on every node of the cluster).

- Configure to run applications as non-root users

- Fluentd acts as the logging aggregator for log events from FluentBit.
- FluentBit and FluentD communication could be configured for secure communication (mTLS) – use of Service Mesh is the preferred choice.
- ElasticSearch is for a centralized log data indexing and storage.
- Kibana is used for log data visualization.