Auditing, Logging, and Observability
Cloud Security Part II
More (harder) Cloud Security Best Practices
Open Policy Agent

- Enforce security policies on cloud deployments when using IaC
  - e.g., “ensure that S3 buckets are not exposed to the public”

- Can be embedded in deployment pipelines or IaC state management systems (e.g., Terraform Cloud)
Example: Open Policy Agent

```haskell
fail contains msg if {
    buckets := [bucket | bucket := input.Resources[_]; bucket.Type == "AWS::S3::Bucket"]
    configs := buckets[_].Properties.PublicAccessBlockConfiguration
    not checkBucketRestricted(configs)
}

msg := sprintf("S3 buckets should block public access", [])
```

From the Assignment 2 autograder
Cloud Security Products

- **Cloud Security Posture Management (CSPM)**: software that scans cloud resources and IaC to spot misconfigurations and insecurities

- **Cloud Native Application Protection Platform (CNAPP)**: CSPM plus more active application monitoring (e.g. with agents on compute resources)
Observability
**Observability** is the ability to ask arbitrary questions about a system without having to know ahead of time what to ask.
Why is observability important?

*Print statements (plus more) for deployed applications*

- **Debugging**: If “something” in your deployed application doesn’t work
  - Where in the chain did something go wrong?
  - Isolate the behavior of the failed component + potential logical assumptions surrounding it

- **Performance**: If “something” in your deployed application feels slow
  - Profile the slowest components of the application, to know where optimizations are needed

- **Security**: If an attacker was able to exploit “something” in your application
  - Where did the exploit originate?
  - How much damage was the attacker able to do?
Logging
Logging from an Application Perspective

**Application event logs**
- For each web request, trace:
  - Handoffs between services
  - Control flow abnormality
  - Errors, exceptions, warnings
- Log levels:
  - DEBUG
  - INFO
  - WARNING
  - ERROR
- Generated by application itself

**Access and security logs**
- For each web request, trace:
  - IP address of client
  - Path requested
  - Response status
- Usually generated by underlying web server or load balancer
### Example: Application Event Logs

<table>
<thead>
<tr>
<th>Timestamp (UTC-08:00)</th>
<th>Message</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:26002 - &quot;GET /api/v1/users/profile/37878c93-e20f-46bf-b046-4d8d6ac28e8b HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:26008 - &quot;GET /api/v1/users/profile/a595dfb2-3402-4369-9009-5a0b259e430f HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.4.146:31826 - &quot;GET /api/v1/users/profile/ba88f5de-e05-4028-a224-169fb3c8046 HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:26008 - &quot;GET /api/v1/users/profile/d4f3e007-edf0-4591-addd-7a766c6729c HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.4.146:11860 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:25992 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.4.146:31826 - &quot;GET /api/v1/feed/latest?before=2024-02-05T16:16:64Z&amp;after=1970-01-01T00:00:00Z HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 127.0.0.1:43920 - &quot;GET /api/v1/health HTTP/1.1&quot; 307 Temporary Redirect</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.4.146:59258 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:26542 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 127.0.0.1:51926 - &quot;GET /api/v1/health HTTP/1.1&quot; 307 Temporary Redirect</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:57 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:45390 - &quot;GET /api/v1/health HTTP/1.1&quot; 307 Temporary Redirect</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:56 (UTC-8:00)</td>
<td>INFO: 10.0.5.56:45378 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:56 (UTC-8:00)</td>
<td>INFO: 10.0.4.146:31028 - &quot;GET /api/v1/health HTTP/1.1&quot; 200 OK</td>
<td>yoctogram-app-container</td>
</tr>
<tr>
<td>February 7, 2024 at 10:55 (UTC-8:00)</td>
<td>INFO: 127.0.0.1:35890 - &quot;GET /api/v1/health HTTP/1.1&quot; 307 Temporary Redirect</td>
<td>yoctogram-app-container</td>
</tr>
</tbody>
</table>

---

**Note:** The timestamps are in UTC-08:00.
Example: Access and Security Logs
Application Log Management

ELK (Elastic) stack, open-source*: Elasticsearch, Logstash, Kibana

1. Logstash ingests incoming application logs
2. Elasticsearch allows easy searching and analytics of logs
3. Kibana helps create visualizations from logs
Elastic Licensing Drama

- In April 2021, Elastic (ELK stack parent company/developer) relicensed Elasticsearch and Kibana from *Apache License 2.0* (open-source) to *Server-Side Public License* (source-available)

- Why this is bad: SSPL forces anyone (e.g. cloud providers) offering ELK as a service to open-source *all* supporting code – which is infeasible

- In response, AWS forked Elasticsearch and Kibana to create OpenSearch, which is still *Apache License 2.0*
Service Logging

● Application logging isn’t always enough

● Sometimes, need visibility into underlying infrastructure to debug
  ○ “Did my web request make it through the load balancer to my container?”

● AWS CloudWatch unifies application and service logs into a single place
AWS CloudWatch

- Log aggregation service for AWS resources
- Each resource forwards logs to a log group
  - Both application and service logs
- Logs are sharded into log streams
  - Representing log events from same logical source – e.g. individual containers
Pros and Cons of CloudWatch

Pros:
● Unify AWS application and service logging in the same place
● Integrate with other AWS services for alarms and visualizations

Cons:
● UI makes it difficult to trace individual events and find issues
● Pricing
CloudWatch Pricing

- Ingestion: $0.50/GB
  - Footgun: this gets charged at raw data size, even when the data is transmitted compressed!

- Retention: $0.03/GB

- Querying: $0.005/GB scanned

This gets expensive when dealing with many resources all logging to CloudWatch.
Security Information and Event Management

- Log management plus network information collection with a security focus
- Anomaly detection to find and alert to potential security events like intrusions
Using Logs in Practice

- Goal: Isolate the source of the problem by understanding where it is not

- Possible methods:
  - Filter logs to only those of the affected users
  - Identify the component causing the issue; use logs to discover which parts of the pipeline are working properly
  - Use your intuition to identify why the problem is occurring

*Logs usually don't tell you what's going wrong directly – but they yield important context.*
Using Logs in Practice

*Error resolution scenario:* some users are unable to access the website
Using Logs in Practice

Security incident scenario: you find a big bill and unknown ECS clusters created
Metrics & Monitoring
Motivation

- Proactively and reactively observe the state of a deployed system
  - To know what changes may need to be made for continued reliability
  - To anticipate future demand and scaling

- Goal: decrease **mean time to recovery** – the time it takes to return to normal operation following an incident
  - Alert to start the incident response process as soon as an issue is detected
What should be monitored?

- **Latency**: how long does it take to service a request?

- **Traffic**: how much demand is being placed on the system?

- **Errors**: what requests fail and why, and context surrounding failures

- **Saturation**: how much demand are compute and storage resources under?
  - e.g. CPU & memory usage, I/O saturation

*Google SRE – The Four Golden Signals, 2017*
Metric Granularity

- Metric collection windows are contextual
  - CPU load should be observed at ~seconds frequencies: utilization spikes don’t last long
  - But probing for storage saturation or web server errors can be less frequent

- Overcollecting metrics can be costly!
Tracing

- **Motivation**: like a stack trace for distributed processes, with performance profiling
- This gives you more details and context around both errors and latency events
Prometheus & Grafana

Open-source metrics collection and management.
Comprehensive Observability Platforms

Integrate logging and metrics into one platform – mostly a commercial space
Why is observability so expensive?
Next Lecture: Serverless Compute (2/12)