Large Scale Security Systems

Project Part 1 - Detection and Active Response

Marcos Caramalho (114834, University of Aveiro)

2025-01-17

Contents

1 Introduction

Large-Scale systems are becoming increasingly in use today's technological landscape. This is mostly because of the mass adoption we've seen of technology as a whole, and as such, systems need to be ever more scalable and resilient. One of the issues we face right now, is with security; systems that are increasingly more complex, become increasingly more prone to some types of attacks and malicious actors constantly try to take advantage of this.

Because of scalability and security, a whole framework of designing these large-scale systems has been adopted. We're still learning as a whole on what works best, but the common opinion is to have a multi layered approach, meaning we can spot strange activity at multiple stages within the system.

This project focuses on implementing a secure, scalable service-oriented architecture capable of detecting and responding to various security threats automatically.

Every asset for this project can be found in the following git repository, under **project-1**: https://github.com/rezzmk/slsa-poc. A single git repository was chosen for **project-1** and **project-2** because part 2 of the project is essentially extending upon what was done for part 1, however, it was asked to keep them separated, thus the distinction.

> Note: The entire infrastructure built for this project is under docker containers. In this document, that fact will be ommited from now on but it's pertinent to keep that in mind, as in, mostly everything is running in a containerized way, which is by itself a plus for scalability purposes, think AWS ECS with CloudFormation, with auto scaling, etc…

1.1 Project Overview

Modern distributed systems require sophisticated security measures that can scale alongside the infrastructure they protect. The implementation shown in this project addresses this need through a comprehensive security monitoring and response system. The architecture combines service-oriented principles with modern security practices to create a robust, scalable solution capable of detecting and mitigating four major types of cyber threats: DDoS (Distributed Denial Of Service), Network Scanning, Spam, and APTs (Advanced Persistent Threats).

1.2 Key Challenges In Large-Scale Systems

The primary challenge in securing large-scale systems lies in their distributed nature. Unlike traditional monolithic applications, distributed systems present multiple attack surfaces and potential points of failure. The scale of monitoring required presents significant technical challenges, particularly in terms of data collection and real-time processing.

Monitoring at scale introduces substantial complexity. When dealing with multiple services generating metrics and logs simultaneously, the system must efficiently collect, process, and store this data while maintaining the ability to respond to threats in real-time. This becomes particularly challenging as the number of services grows and the volume of monitoring data increases exponentially.

The distributed nature of the system also complicates security responses. When a threat is detected, the system must coordinate responses across multiple services and infrastructure components. This coordination must be both timely and accurate, ensuring that security measures are applied consistently across the entire system without creating new vulnerabilities or disrupting legitimate service operations.

1.3 Scope and Limitations

This implementation focuses on creating a practical, scalable security solution while acknowledging certain operational constraints. The system includes:

- A distributed service architecture comprising multiple independent services, each contributing to the overall functionality while maintaining its own security boundary (independent services). This architecture allows for horizontal scaling while ensuring that security measures scale proportionally with the system.
- A centralized monitoring and alerting system serves as the cornerstone of our security infrastructure. This system collects and analyzes data from all services, providing a comprehensive view of the system's security status while maintaining the ability to process high volumes of monitoring data efficiently.
- Automated threat detection and response capabilities, enabling the system to react to security threats without constant human intervention. This automation is crucial for maintaining security at scale, where manual responses would be impractical or too slow to be effective.

2 State of The Art

Recent advances in cloud computing and distributed systems have driven significant innovation in security monitoring and response systems. This section examines current approaches and technologies that inform our implementation, alongside the course materials taught in the class this project is scoped too.

2.1 Modern Monitoring Approaches

2.1.1 Prometheus Ecosystem

The Prometheus ecosystem represents one of the current state-of-the-art in metrics-based monitoring for cloud-native environments. Its pull-based architecture offers significant advantages in terms of reliability and scalability compared to traditional push-based systems. The ecosystem's components work together to provide a comprehensive monitoring solution:

- The Prometheus server handles metrics collection and storage, employing a sophisticated time-series database optimized for monitoring data. Its pull-based architecture ensures that metrics collection continues even if individual targets become temporarily unavailable, making it particularly suitable for dynamic environments where services may come and go frequently.
- PromQL, Prometheus's query language, provides capabilities for metric analysis and alert definition. Its ability to perform complex calculations and aggregations across multiple metrics makes it particularly valuable for detecting subtle patterns that might indicate security threats.
- Using these metrics and alongside alerting, we can make sure these alerts end up getting acted upon, via some active response measures, when needed. ### Security Information and Event Management (SIEM)

In modern distributed architectures, SIEM systems play a crucial role in security monitoring and incident response. This implementation leverages Wazuh, a modern open-source SIEM solution. Unlike traditional SIEM systems that focus solely on log analysis, Wazuh provides a comprehensive security platform that combines multiple security functions.

Wazuh's architecture enables real-time threat detection through log analysis and correlation rules. The system processes security events from multiple sources, including system logs, application logs, and network traffic data. This multi-source approach provides a more complete security picture than single-source monitoring solutions.

We are also able to rely on its active response capabilities to react on certain alerts. These active responses can be customized to our needs.

2.2 Attack Detection and Prevention

2.2.1 DDoS Detection

Modern DDoS detection approaches have evolved beyond simple rate-based detection. The current best practices employ a multi-layered approach that combines multiple detection methods:

- Rate-based detection serves as the first line of defense, monitoring incoming request rates and comparing them against established baselines. However, sophisticated DDoS attacks often try to stay below rate-based thresholds. Therefore, pattern recognition is something increasingly important, allowing systems to identify attack signatures that might not trigger alerts from the beginning
- Resource utilization monitoring provides another crucial layer of detection. By monitoring system resources like CPU, memory and network bandwidth, systems can detect attacks that might not be apparent from request rates alone. This is useful for application-layer DDoS attacks that might use seemingly legitimate requests to exhaust system resources.
- Behavioral analysis represents the most advanced form of DDoS attacks detection. By establishing baselines from normal usage of the system, it is possible to detect deviations from this baseline, potentially finding an attack that's evading traditional denial of service methods.

2.2.2 APT Detection

Modern APT detection requires a sophisticated approach that can identify the multiple stages of an advanced attack. Current technology focuses on:

- Multi-stage attack recognition capabilities that can correlate seemingly unrelated events over time. This is crucial for identifying APTs, which often operate over extended periods and may use multiple attack vectors.
- Behavior-based analysis systems monitor for unusual patterns in system and user behavior. This might include unusual access patterns, unexpected data transfers, or anomalous system configurations. Modern systems use machine learning models to establish baselines of normal behavior and identify significant deviations.
- Network traffic analysis plays a vital role in detecting command and control (C2) communications. Modern systems analyze traffic patterns, DNS queries, and encrypted traffic metadata to identify potential C2 channels, even when the actual traffic content is encrypted.

2.2.3 Network Scanning Detection

Modern network scanning detection has evolved significantly beyond simple port monitoring. Current technologies employ sophisticated detection mechanisms across multiple layers:

- Pattern-based detection serves as the foundation of modern scanning detection. Systems analyze request patterns across services, looking for telltale signs of reconnaissance activities. This includes monitoring for sequential access attempts, unusual HTTP methods, or systematic probing of API endpoints. Modern implementations particularly focus on detecting distributed scanning attempts, where attackers might use multiple source addresses to evade detection.
- Rate limiting has evolved to become more context-aware. Rather than implementing simple request-per-second limits, modern systems consider factors such as the ratio of successful to failed requests, the diversity of endpoints being accessed, and the historical behavior of clients. This contextual approach helps distinguish between legitimate high-volume access and malicious scanning activities.
- Behavioral fingerprinting represents the cutting edge of scan detection. By analyzing how clients interact with services - including factors like request timing patterns, header variations, and response handling - systems can identify automated scanning tools even when they attempt to mimic legitimate traffic. This approach is particularly effective against sophisticated scanning tools that implement rate limiting and randomization to avoid detection.
- Honeypot integration provides an additional layer of detection capability. By strategically placing honeypot endpoints within the service architecture, systems can identify scanning activities early in the reconnaissance phase. Modern implementations often use dynamic honeypots that adapt their behavior based on the detected threat patterns.

2.2.4 Anti-spam Technologies

Anti-spam technologies have advanced from simple keyword filtering, employing multi-layered approaches to combat increasingly sophisticated spam attacks:

- Content analysis represents the initial layer of defense, with modern systems like SpamAssassin using sophisticated rule sets and machine learning algorithms to evaluate message content. These systems analyze not just text content, but also examine email structure, embedded links, and attachments. The analysis encompasses both traditional spam indicators and emerging threat patterns, allowing for detection of new spam campaigns.
- Reputation systems have become increasingly sophisticated, moving beyond simple IP-based blacklists. Modern systems maintain comprehensive databases of sender reputation, taking into account factors such as:

  - Historical sending patterns
  - Authentication records
  - Message engagement metrics
  - Network behavior
  - Domain age and reputation

- Pattern matching has evolved to incorporate machine learning and statistical analysis. Modern systems can identify spam campaigns by recognizing subtle patterns across multiple messages, even when individual messages might appear legitimate. This includes analysis of:

  - Message timing patterns
  - Content similarities
  - Sending infrastructure patterns
  - URL and domain patterns
  - Image analysis for graphical spam

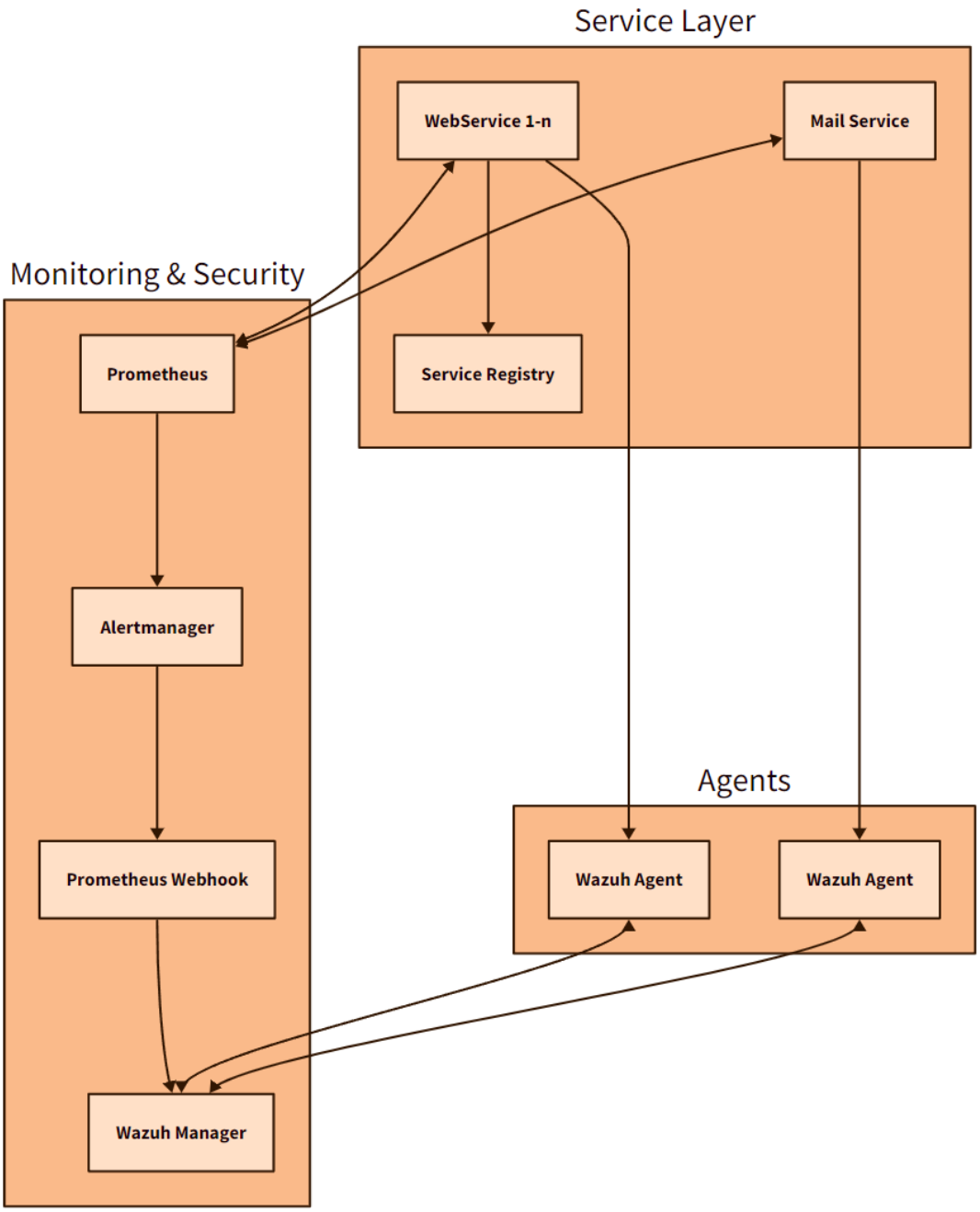
- Machine learning approaches represent the SoA of spam detection. These systems can:

  - Adapt to new spam techniques in real-time
  - Identify sophisticated phishing attempts
  - Detect context-aware spam that might bypass traditional filters
  - Learn from user feedback and behavior
  - Correlate threats across multiple organizations

The integration of these technologies creates a robust defense against both mass spam campaigns and targeted spamming attempts. Modern systems particularly excel at identifying and blocking sophisticated attacks that combine spam with other threat vectors, such as phishing or malware distribution.

3 Service Architecture

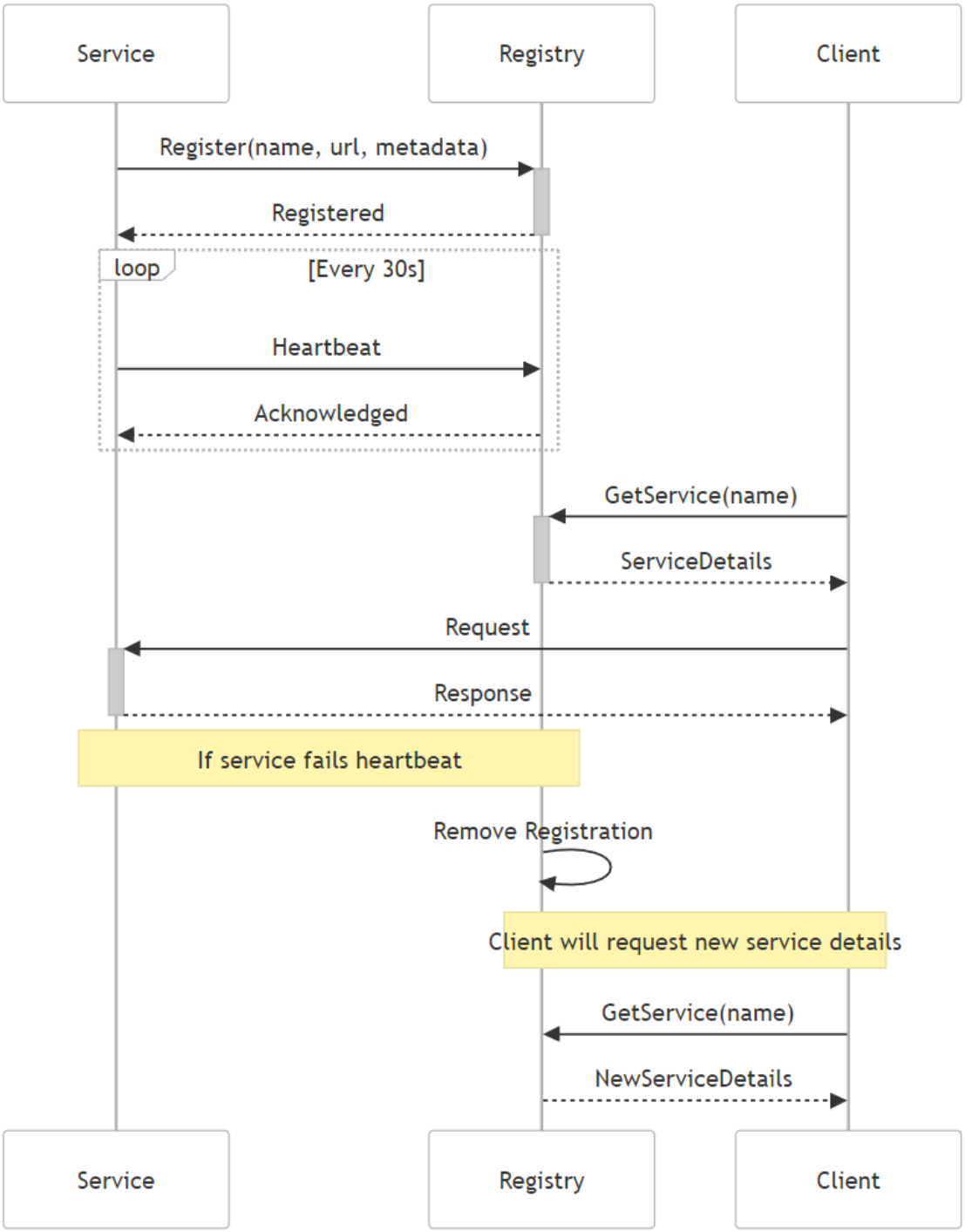
To work on this project, the following architecture was chosen (Note: everything in here runs in docker):



At a high level, we have N services (3 while testing), a mail service to PoC spam detection, as as well one Wazuh agent per one of these services. Additionally, we have Prometheus for metrics collection, Alertmanager to dispatch the alerts fired on prometheus to the respective endpoints, mainly the prometheus webhook, as well as Wazuh, which reads these alerts as logs and acts accordingly.

3.1 Service Registry and Discovery

One of the architectural requirements was to work on a service discovery and registry system, mostly using a separate registry that can keep track of these other services. To achieve this, a Service Registry module was created, in .NET, that has some endpoints. The registration and discovery flow can be seen in the next diagram:

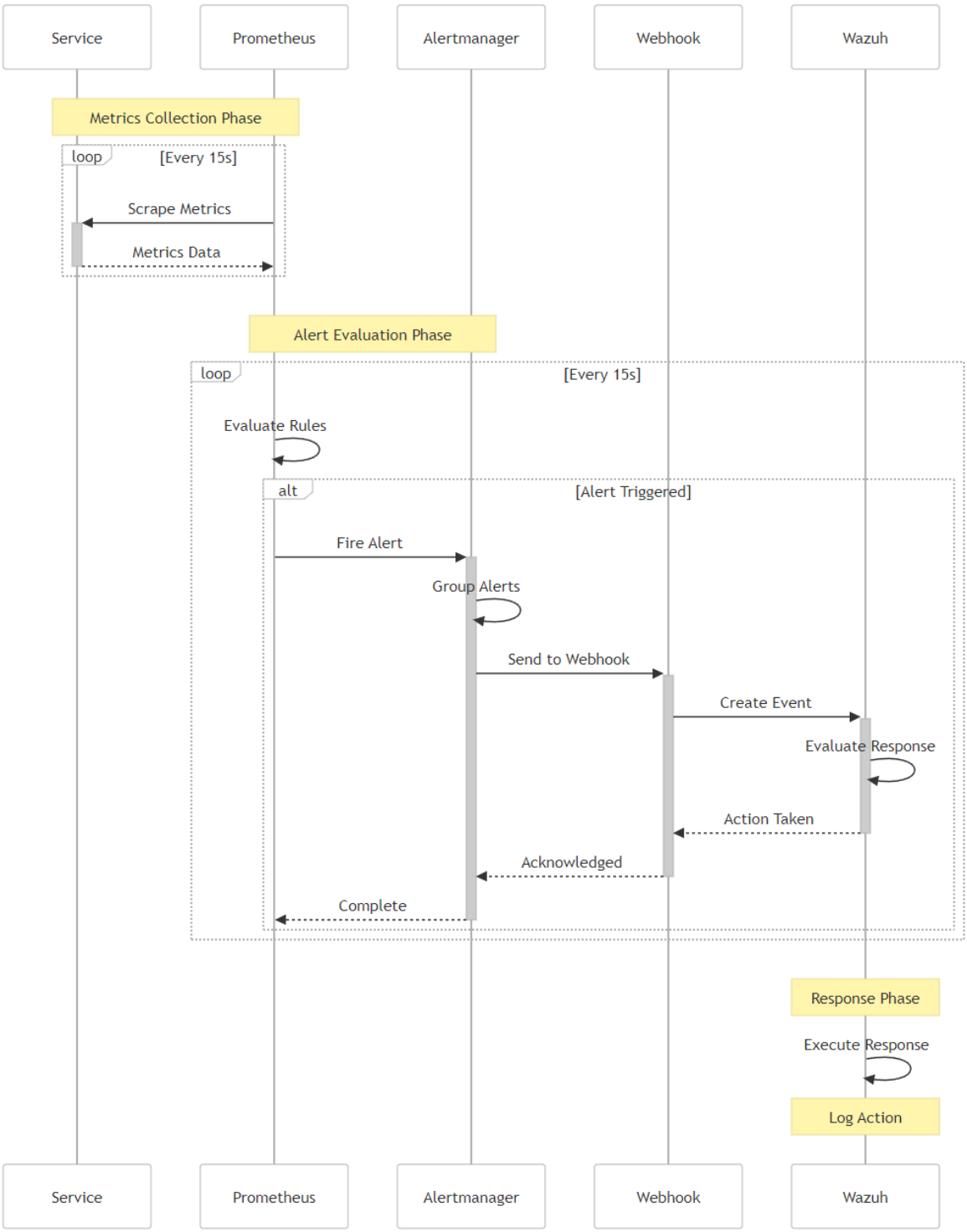


At a high level, every time a service is started, it joins against the registry by calling a RESTful endpoint on the Service Registry. Discovering existing services can be done in this same service by using the provided REST API. ## Monitoring Infrastructure

In order for the Wazuh SIEM to get alerts from Prometheus, there are a few options. The way that was found to work the best for our scenario, was to create a webhook that will be called when Alertmanager gets an event (Prometheus “Firing” state).

For instance, detecting a DDoS attack is possible because Prometheus is calling into the metrics endpoint of these services. When it detects a flood of requests, it triggers an alert to alertmanager, which in turn calls a webhook. This webhook will write the alert in a custom format to a specific log file that’s read by every agent. These log files are monitored by Wazuh so it knows given the rules and custom decoder for the logs, to throw a new event of a given criticality.

Below is the complete high level flow for the event monitoring:



The monitoring infrastructure forms a critical component of this project's security architecture. It provides visibility, while enabling automated response to security threats. ## Implementation Details

3.1.1 Service Components

Each service in our architecture includes:

- RESTful API endpoints
- Prometheus metrics middleware
- Service discovery client
- Wazuh agent integration
- Logging infrastructure

3.1.2 Monitoring Stack

The monitoring stack consists of:

- Prometheus server for metrics collection
- Alertmanager for alert routing
- Custom webhook for alert processing
- Wazuh manager for security event handling

3.1.3 Security Components

Security is implemented through:

- Wazuh agents on all services
- Custom active responses
- MTD controller for service rotation
- Alert correlation engine

4 Risk Management and Mitigation

Risk management in large-scale distributed systems requires a deep understanding of potential threats and their impact. The approach taken here, combines traditional risk assessment methodologies with modern security practices targeting distributed architectures.

4.1 Attack Surface Analysis

A thorough understanding of our attack surface reveals multiple potential entry points that require specific protection measures. Our analysis identifies three primary attack surface categories:

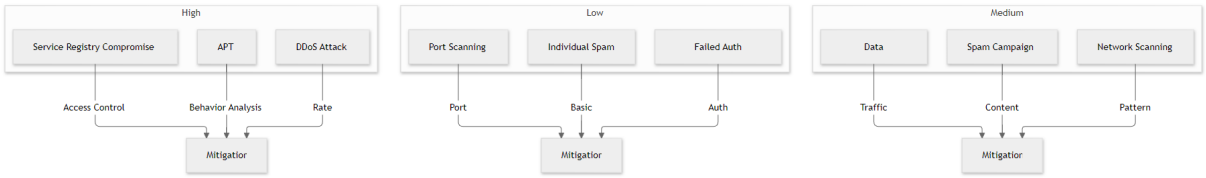
Network Layer Exposure represents the most immediate attack surface. Public-facing services are particularly vulnerable to attacks and reconnaissance attempts. Our service discovery mechanism, while essential for system operation, also presents a potential target for attackers attempting to manipulate service routing. Inter-service communication channels, though internal, must be protected against potential lateral movement by attackers who gain a foothold in the system.

Application Layer vulnerabilities present more sophisticated attack vectors. Authentication systems must protect against both brute force attempts and credential theft while maintaining usability. API endpoints require protection against various attack types, from simple flooding to complex injection attempts. The service registry, being central to system operation, requires particular attention to prevent manipulation that could affect service routing and discovery.

Infrastructure Layer Risks stem from the underlying systems supporting our services. Container runtimes must be protected against escape attempts and resource exhaustion. Monitoring systems, while essential for security, must themselves be secured against tampering that could blind us to attacks. Log management systems require protection against both tampering and resource exhaustion from log flooding attacks.

4.2 Risk Assessment Matrix

At a high level, all possible attacks that are thought of in this system will end up in a mitigation stage, provided they're successfully detected. A diagram for this can be found next:



More context on how we mitigate or think of mitigating the attacks can be found in the next section for Mitigation Strategies

4.3 Mitigation Strategies

The mitigation strategies chosen for this project are ones that are easily implementable and by no means the best that can be done. Things like rate-limiting and behavioral detection aren't implemented at all, but as a PoC, below are the mitigation strategies for each attack type ### DDoS

- Detection through Prometheus metrics (monitoring request rates)
- IP blocking via Wazuh active responses (firewall-drop) ### Network Scanning

- Detection through 4xx error pattern monitoring
- IP blocking for detected scanning behavior via Wazuh

4.3.1 Spam

- Content analysis using SpamAssassin
- Sender blocking through Postfix after detection
- Prometheus metrics for spam monitoring ### APT
- Multi-stage attack detection through prometheus rules
- C2 detection through beaconing patterns
- Integration with Wazuh for responses

Potential improvements here would be to implement rate-limiting, as stated before, load balancing, dynamic resource scaling, MTD, request throttling, and so on…

5 Threat Detection Implementation

The attacks covered in the project can be detected via alerts that are triggered given certain Prometheus metrics. This makes the whole solution a bit easier as we can always keep adding new rules based on existing or new metrics implemented as the system scales and evolves.

Responding actively to these threats is mostly done with active response scripts within wazuh.

5.1 DDoS Attack Detection

The DDoS detection implementation relies on Prometheus metrics and alerts combined with Wazuh responses.

The core detection rule in Prometheus monitors request rates:

```
- alert: HighRequestRate
  expr: sum by (instance, client_ip) (
    rate(http_requests_total{job="dotnet-webservices"}[1m])
  ) > 100
  for: 30s
  labels:
    severity: critical
    attack_type: ddos
```

When this alert triggers, the webhook forwards it to Wazuh, which executes the following response (firewall block):

```
<!-- Wazuh Active Response -->
<command>
  <name>firewall-drop</name>
  <executable>firewall-drop</executable>
  <timeout_allowed>yes</timeout_allowed>
</command>

<active-response>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>200001</rules_id>
  <timeout>180</timeout>
</active-response>
```

This implementation can be tested in multiple ways, `bonesi` is one of them, although simpler approaches can also be taken by using stress test tools such as `ab` (Apache benchmark). ## Network Scanning Detection

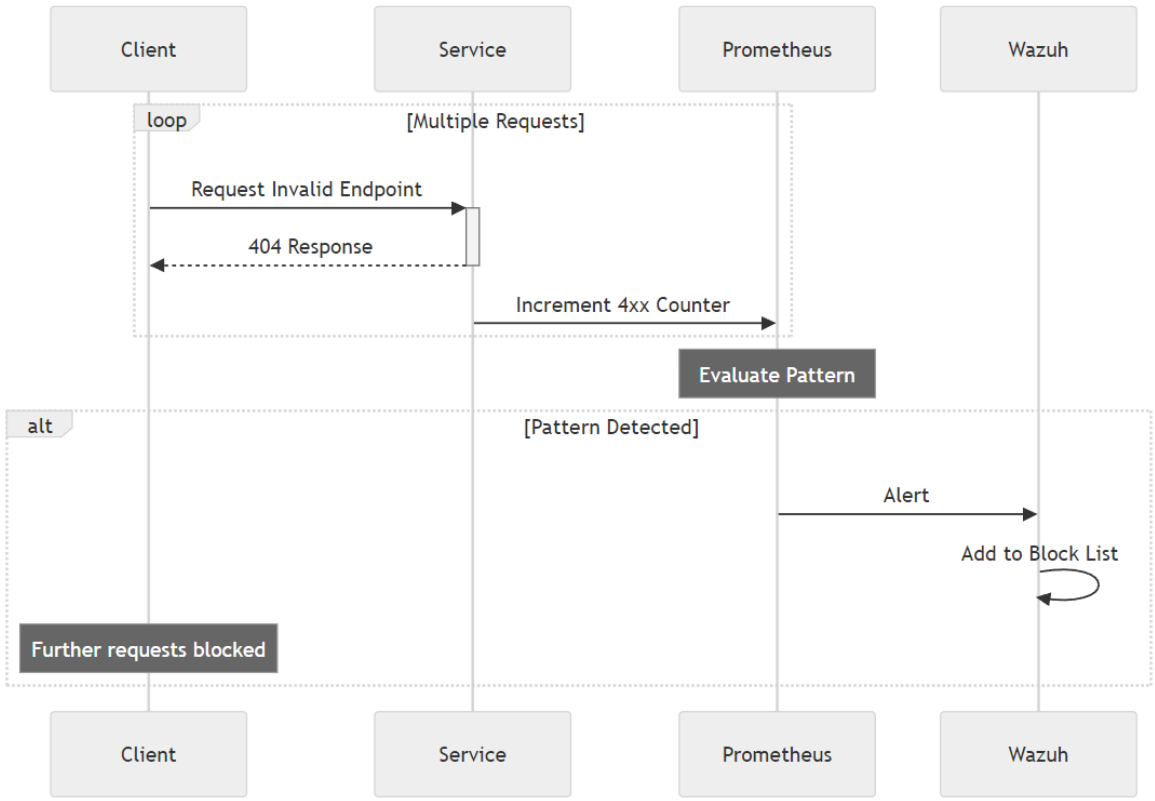
Scan Detection, in this project, is done by identifying patterns of 4xx responses that typically indicate scanning behavior. The system uses the following prometheus main rule:

```
- alert: NetworkScanningDetected
  expr: |
    sum by (instance, client_ip) (
      rate(http_requests_total{status=~"4..", job="dotnet-webservices"
          }[1m])
    ) > 2
  for: 10s
  labels:
    severity: warning
    attack_type: network_scan
    attack_subtype: api_scanning
```

On the web services, some testing endpoints, purely for this PoC, were created in order to mimic scanning. For instance, the following `/admin` endpoint was created:

```
[HttpGet("admin")]
public IActionResult Admin() {
    if (random.NextDouble() < 0.9) {
        return Unauthorized();
    }
    return Ok(new { message = "Admin area" });
}
```

The detection flow for network scanning, as is implemented in this project, can be viewed at a high level in the diagram below:



5.2 Spam Attack Detection

Spam detection combines SpamAssassin with custom monitoring. The implementation includes:

1) SpamAssassin configuration

```
# Custom rules in local.cf
body BITCOIN_SCAM /bitcoin|cryptocurrency|wallet address/i
score BITCOIN_SCAM 3.0

body URGENCY /urgent|immediate action|account.*suspend/i
score URGENCY 2.0
```

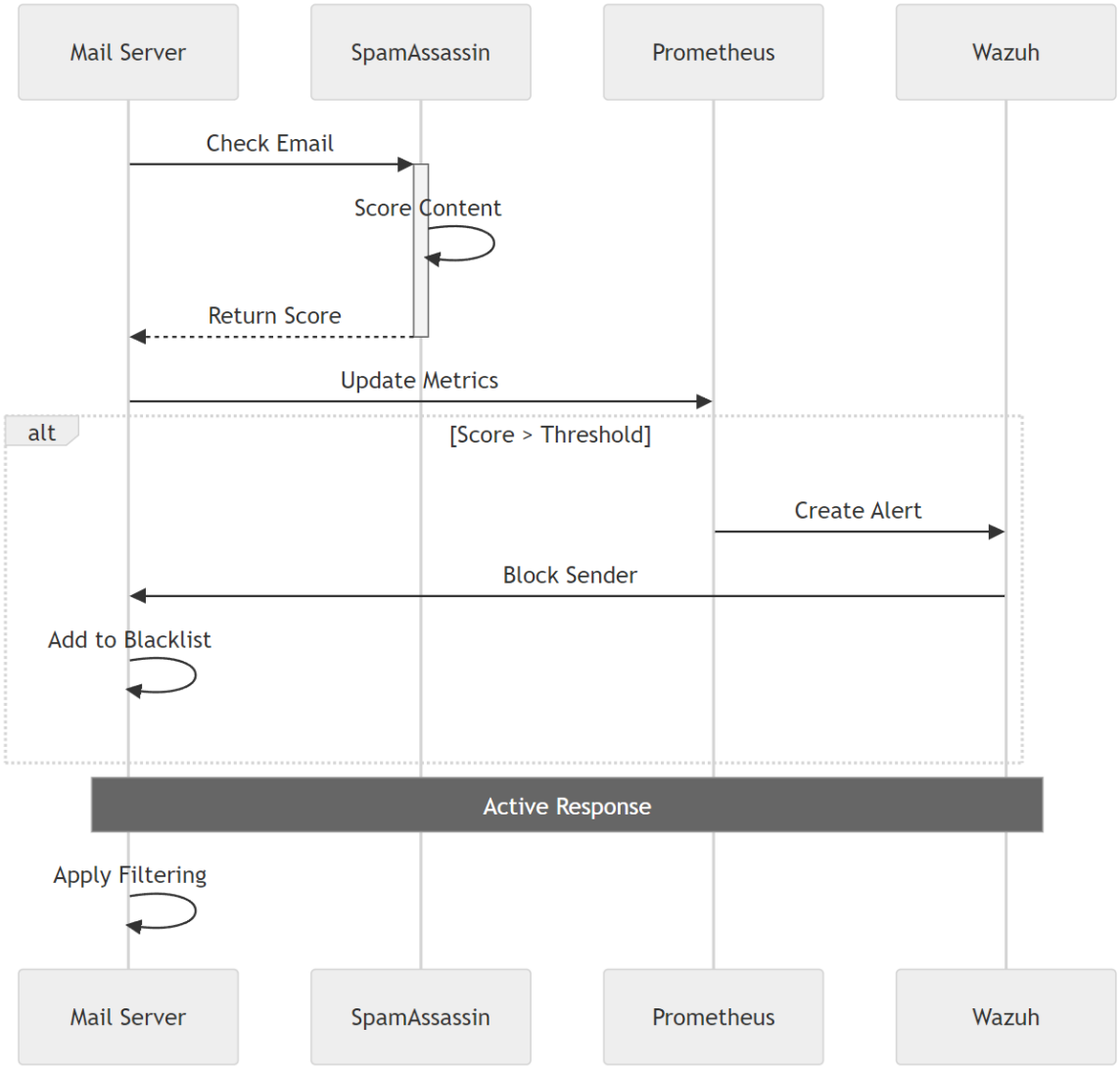
2) Prometheus metrics collection:

```
SPAM_MESSAGES = Counter('spam_messages_total', 'Total spam detected')
SPAM_SCORE = Gauge('spam_score', 'SpamAssassin score', ['sender'])
```

3) Wazuh Integration for blocking repeat offenders. Full disclosure, the current implementation is not actually employing this script yet, but the whole infrastructure and alerts work.

```
<rule id="200011" level="12">
    <if_sid>200010</if_sid>
    <frequency>5</frequency>
    <timeframe>300</timeframe>
    <description>Multiple spam attempts from same source</description>
</rule>
```

The high level detection flow for spam can be seen below:



5.3 APT Detection

The APT detection focuses on identifying different stages of attacks through behavioral patterns:

1) Recon Detection (Prometheus):

```
- alert: SuspiciousAuthActivity
  expr: |
    sum by (instance, client_ip) (
      rate(http_failed_auth_total{job="dotnet-webservices"}[1h])
    ) > 0.1
```

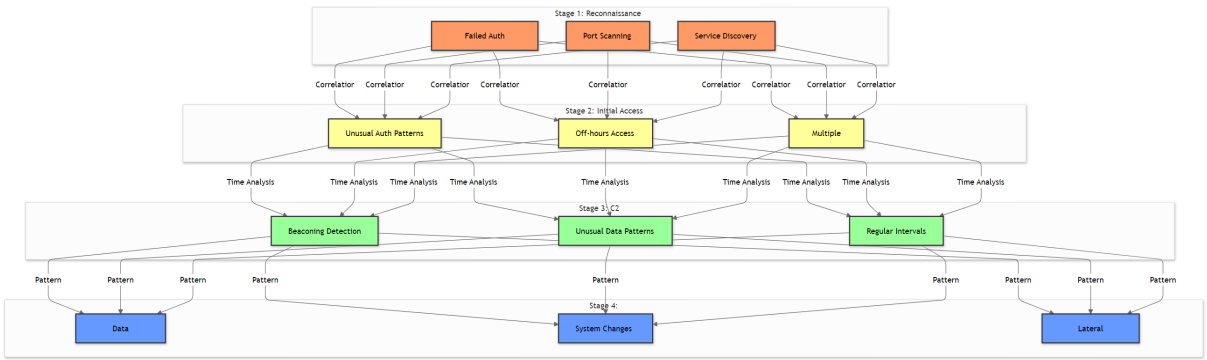
2) C2 Detection (Prometheus):

```
- alert: PotentialC2Traffic
  expr: |
    (
      rate(http_requests_total{status=~"2.."}[30m]) > 0
    )
    and
    (
      rate(http_requests_total{status=~"2.."}[30m]) < 0.1
    )
```

3) Data Exfiltration Detection (Prometheus):

```
- alert: PotentialDataExfiltration
  expr: |
    rate(http_response_bytes_total{job="dotnet-webservices"}[5m]) >
        102400
```

Essentially it was assumed that an APT attack would be done in multiple stages. Although not everything was implemented, a potentially more robust solution would be something like the following diagram:



It shows a stage 1 of recon that can turn into an stage 2 initial access. Once access has been achieved, a command and control stage can start, where data exfiltration and the like can happen. # Solution Evaluation

This implementation's evaluation focuses on practical observations and identified limitations of the current system.

5.4 Current Implementation Capabilities

The system successfully demonstrates:

1) Attack Detection:
    1) DDoS: Detection through request rate monitoring and automatic response through Wazuh
    2) Network Scanning: Pattern recognition of 4xx errors with IP blocking
    3) Spam: Integration of SpamAssassin with Prometheus metrics and Wazuh responses
    4) APT: Multi-stage detection focusing on authentication patterns and C2 behavior
2) Monitoring Integration:
    1) Successful integration between Prometheus and Wazuh
    2) Functional Alert routing through Alertmanager and webhook
    3) Metric collection from multiple services
    4) Log aggregation and analysis
3) Response Automation:
    1) Automatic IP blocking via firewall, for detected threats
    2) (Not Completed but infra done) Spam sender blocking at mail service level
    3) Alert correlation between different detection systems
    4) Active response execution through Wazuh agents

5.5 Limitations

It's important to acknowledge the current limitations of the implementation:

1) Scaling Limitations
    1) Single Prometheus instance without federation
    2) No high availability for critical components
2) Detection Capabilities
    1) Basic DDoS detection without traffic analysis
    2) Simple pattern matching for scan detection
    3) Limited APT detection sophistication
    4) Basic spam detection rules
3) Response Mechanisms
    1) Limited to IP blocking and sender blocking

2) No gradual response escalation
3) No automated recovery problems
4) Single Wazuh instance

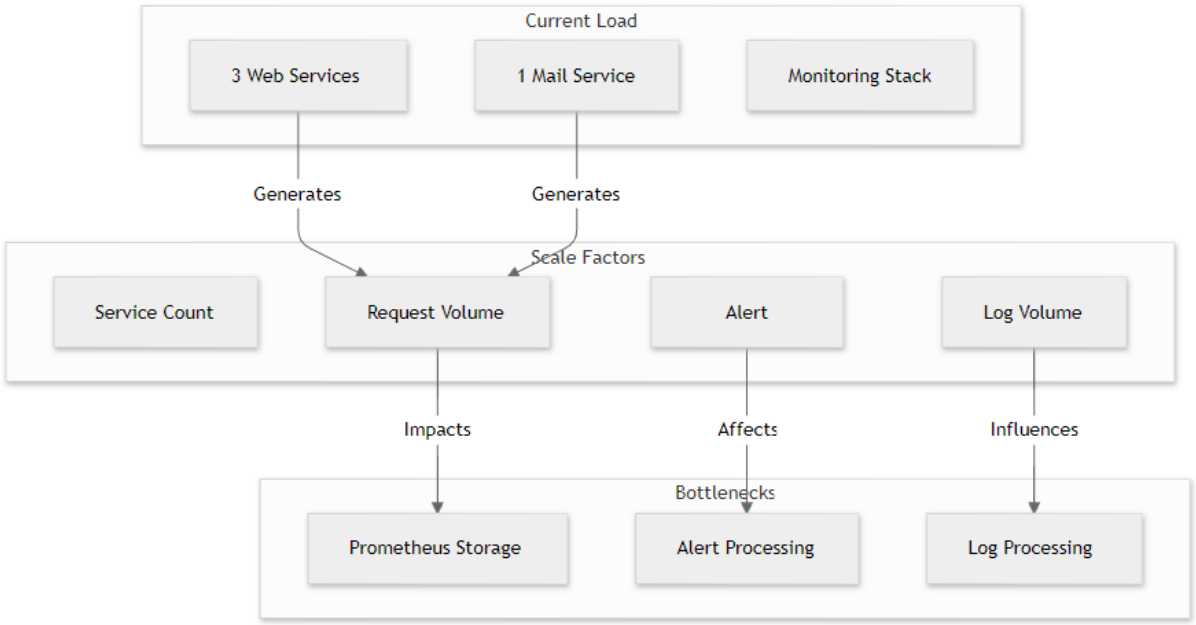
5.6 Performance Considerations

To evaluate performance, the current implementation shows:

1) Detection Times
    1) DDoS: Detection within 30 seconds of pattern start
    2) Scanning: Detection after ~10 seconds of 4xx errors
    3) Spam: Immediate per-message analysis
    4) APT: Variable, depending on pattern matching
2) Resource Usage
    1) Prometheus storage grows linearly with service count
    2) SpamAssassin analysis adds a delay per email processing, ~2s from testing
    3) Wazuh agent overhead is minimal on services
    4) Webhook processing shows occasional latency

5.7 Scalability

The scalability of the system can be seen in a zoomed out way with the next diagram:



It essentially shows the bottlenecks are mostly in terms of log storage and log processing. This can be mitigated with more resources as well as extra instances of Prometheus and Wazuh.

Because the implementation here uses Docker for everything possible, all of these factors would be reduced using some of the existing cloud infrastructures, such as AWS ECS (Elastic Container Service), which can auto-scale given certain load metrics.

6 Conclusions

This project demonstrates both the effectiveness of modern security monitoring approaches and the challenges of implementing them at scale. While the implementation successfully meets the core requirements, it also highlights areas where enterprise-grade solutions would need additional development.

6.1 Summary

- Successfully implemented detection for all required attack types
- Created automated response system
- Integrated multiple security tools
- Demonstrated a scalable architecture

6.2 Future Improvements

6.2.1 High Priority

- Wazuh Cluster Implementation
- Prometheus Federation
- Advanced MTD

6.2.2 Medium Priority

- ML-based Detection
- Enhanced Correlation
- Better APT

6.2.3 Low Priority

- Custom Dashboard

6.3 Lessons Learned

6.3.1 Architecture Decisions

- Service Discovery crucial for scalability
- Monitoring overhead must be considered, something simple can turn into a behemoth that's hard to maintain and impossible to implement in a cost effective way
- Active Response measures need careful tuning

6.3.2 Security Insights

- Multi-layered detection is the most effective way we currently have of dealing with scalable security
- Correlation improves accuracy provided logs are good
- Response automation essential