SP-110 Green Linux User Anomaly Detection System

CS 4850
Fall Semester 2025
Date: 12/2/2025
Instructor: Sharon Perry

Marlowe Elmiger, Miles Lindsey, Tochukwu Okwudire

Website:
https://sp110-green-linux.github.io/sp110-green-linux/
Github:
https://github.com/sp110-green-linux/anomaly-spotter
Statistics
Lines of code: 630
Number of components: 6
Total man-hours: 120
Status: 80%

Table of Contents:

Introduction..4

Version Control..4

Requirements..4

Assumptions and Dependencies..4

Functional Requirements..5

Log Extraction..5

Anomaly Definition..5

Anomaly Identification..5

Nonfunctional Requirements..5

Alert System..5

Log Protection..5

Reliability..5

Performance..6

User Stories..6

Analysis/Design..6

Design Considerations..6

Assumptions and Dependencies..6

General Constraints..6

Development Methods..8

Architectural Strategies..9

System Architecture..10

Detailed System Design..11

Constraints..12

Resources..12

Development..12

Log Gatherer..12

Live Log Gatherer..12

Log Parser..13

Feature Extractor..13

Machine Learning Model..13

Alert System..13

Database Connection - Not applicable..13

Project Setup..13

Test Plan and Report..14

Test Objectives 14

Scope of Testing 14

Test Cases 15

Test Procedures 15

Test Environment 15

Test Data 16

Test Report 16

Summary 16

Appendix 17

References 17

Introduction

Our team aims to create a system that identifies anomalies found in a user's logs within a Linux machine. In order to define what an anomaly is, it is most important to teach this system what "normal" logs may look like, which can lead to an outlier where an anomaly occurs. Some examples of what can be classified as an anomaly might be a high frequency of sudo calls, numerous invalid password attempts, or even logs at times that may be deemed suspicious.There are many safeguards put in place to protect a user's account from others, especially those with malicious intentions. Security, however, should not stop at the assumption that nobody can get past these safeguards. Many factors can lead to an account being compromised by, or the account itself can belong to, a malicious user. Our system will be trained using data sets that have been extensively curated to simulate what can be considered suspicious in a professional environment.

Version Control

Our team used Github to track the progress of our system. Discord was used to communicate, conduct meetings, and allocate tasks to the team members. Through a balance of both applications, our team had the ability to communicate and collaborate in our task of building our system.

Requirements

Assumptions and Dependencies

- Open source datasets
- Python libraries such as scikit-learn, PyTorch, TensorFlow etc.
- There exists the assumption that this system is reading from someone on only the Linux operating system. For this project, we will be using the Ubuntu distro.

Functional Requirements

- Log Extraction

    -The system shall be able to read logs from a user
    -The system shall pull logs from the relevant directories

- Anomaly Definition

    -The system shall understand how to properly define what an anomaly is and what a normal log is

- Anomaly Identification

    -The system shall properly identify anomalies based on comparisons of the anomaly dataset and user logs
    -The system shall follow a series of steps to make an alert when an anomaly is identified

Nonfunctional Requirements

- Alert System

    -There should exist some kind of alert system to let administrators easily monitor when suspicious activity is occurring

- Log Protection

    -The system should keep all logs pulled protected, ensuring nobody is granted access to this information

- Reliability

    -There should be as few false flags as possible, with the amount getting smaller as the system learns

- Performance

  -The system should have the ability to monitor more than one user at a time

User Stories

- As a security administrator, I would like to be alerted when an anomaly occurs.
- As a security administrator, minimal false flag alerts.
- As a security administrator, I would like for the system's accuracy to be easily benchmarked with a simple series of tests.
- As a Linux user, I would like for my relevant data to remain protected.
- As a systems administrator, I would like for the anomaly detection system not to add to any overhead, ensuring smooth operation for the user.

Analysis/Design

Design Considerations

Assumptions and Dependencies

- **Related software or hardware:** The system will need a linux computer with around 16 GB of RAM, and around 100 GB of storage in order to handle large Loghub datasets. Python is needed along with certain machine-learning libraries like PyTorch (To analyze the logs).

- **End-user characteristics:** Users will understand basic security concepts, but are by no means experts. End-users also have a solid grasp of logs.

General Constraints

- **Hardware or software environment:** No special hardware is required for this project. The system should be able to run smoothly on a regular linux computer (Or virtual machine)

- **End-user environment:** This system will generate clear alerts/messages depending on the outcome.

- **Availability or volatility of resources:** Logs will inevitably be messy at times or even incomplete. The system needs to be able to handle this with error handling. It needs to be under the assumption that the data won't be perfect all the time.

- **Standards compliance:** The system must follow the GDPR (General Data Protection Regulation). Protecting users' personal data is imperative. We also want to follow the National Institute of Standards and Technology standards to reduce the chance of cyberattacks.

- **Interoperability requirements:** The system needs to have multiple systems communicate messages and data with the machine running the detection system at one time, so the development must take this into consideration.

- **Interface/protocol requirements:** There must exist communication protocols to drive the alert system and to send the log data to the detection system.

- **Security requirements (or other such regulations):** We want to construct the detection system in a way that adversaries can't steal sensitive information or alter the results after it is run.

- **Memory and other capacity limitations:** Since Loghub datasets tend to be very large, our system must have enough storage and memory to handle the processing of logs effectively. 100 GB of storage and 16 GB of RAM is optimal.

- **Performance requirements:** The program should be able to detect threats with high accuracy. We don't want the user to experience frequent amounts of false positives or false negatives.

- **Network communications:** We want the log data to be encrypted if being sent over a network.

- **Verification and validation requirements (testing):** The system will be tested using fake attacks in order to make sure it works as intended.

Development Methods

- For this project, we will use machine learning to detect unusual patterns in the log files, which could signal anomalies or attacks. One machine learning approach we are considering is an unsupervised learning method such as isolation forest. This would be easier than supervised learning, at least with the logs we are dealing with, because the data doesn't have to be labeled.

- Deep learning models might be effective for this project too. There are some good deep learning Python libraries that would be beneficial.

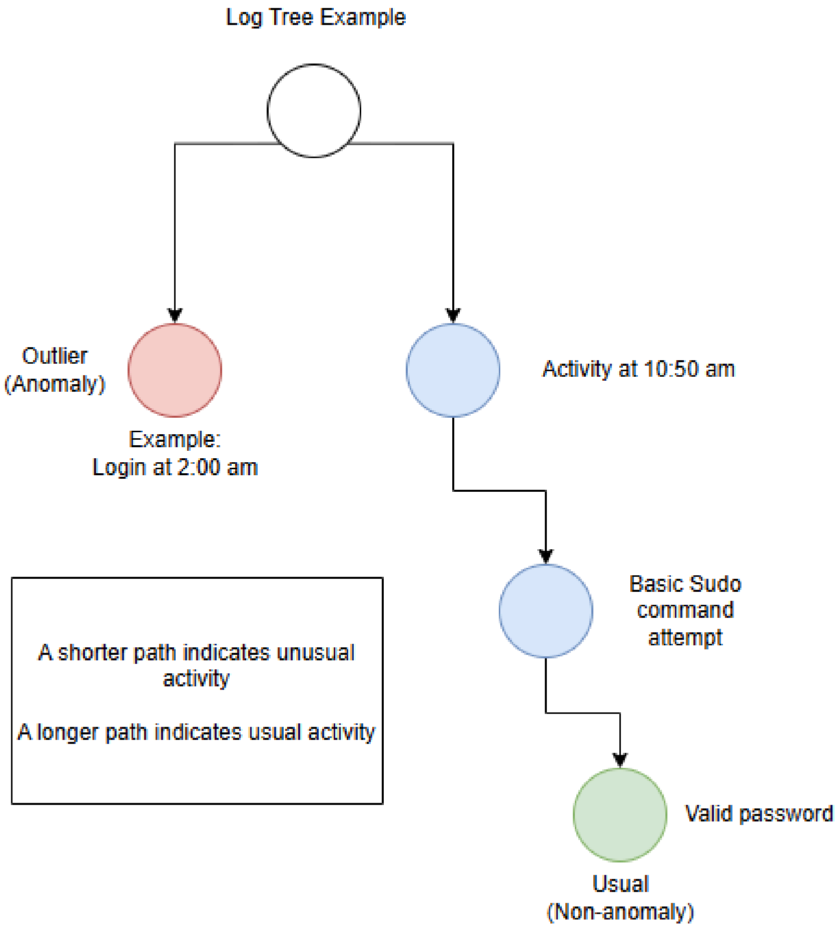

Figure 1. Example of an isolation forest tree

Architectural Strategies

Use of a particular type of product (programming language, database, library, etc. …): The programming language that we will be using for this project is Python. We will also be using several machine-learning and data science libraries alongside it. The dataset/collection of logs we will be using is called Loghub (Available on github). It's open source and is a great choice for this project.

Reuse of existing software components to implement various parts/features of the system: We want to use existing libraries for log parsing and machine learning such as PyTorch and logparser, respectively.

Future plans for extending or enhancing the software: A graphical user interface with the capability for user login and privilege can be crafted creating more detailed, visual information for alerts.

User interface paradigms (or system input and output models): This system will generate clear alerts/messages so that users can take immediate action

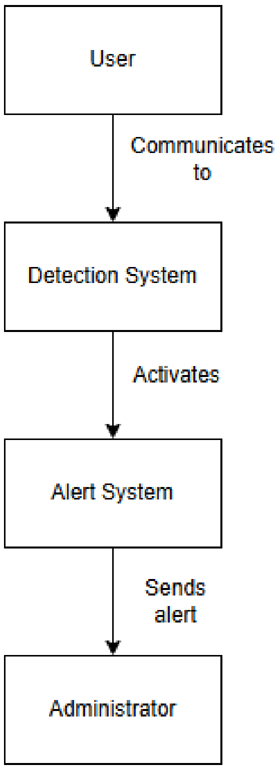

Figure 2. Conceptual model for alert process

External databases and/or data storage management and persistence: Logs will be stored in a database so we will be able to access them anytime

Concurrency and synchronization: Logs to be partitioned individually by workers. Race condition security will be implemented.

Communication mechanisms: The ability to communicate via data streams is made easy by Python. The possibility of using libraries such as Kafka is open.

System Architecture

Component #1 - log gatherer: Collects raw logs.

Component #2 - log parser: splits up and organizes information in a structured way that we can easily use

Component #3 - Feature extractor: uses the information from the logs like error codes and timestamps, and converts it into a numerical value that can be easily learned in machine learning

Component #4 - Anomaly finder: Using machine learning of the logs, anything that looks strange or abnormal will be flagged.

Component #5 - Alert system: When there is an anomaly, the user will be alerted with a message. If there is no anomaly, the user will be notified that nothing seems suspicious

Detailed System Design

Component #1: The purpose of the log gatherer is to reliably take in and process the raw/original logs. Some of the logs to be expected are system logs, application logs, database logs, cloud audit, etc.

Component #2: The purpose of the log parser is to comb through the raw data collection from the log gatherer and organize it into a structured scheme along with tags. The schema that would be made will allow the data to be extremely legible and understandable. The input it receives will be directly from the gatherer.

Component #3: For the feature extractor, the overall purpose is to turn the parsed data logs into machine learning features that are based on numerical and categorical data. Input for this would be the previously parsed data logs. The output for this would be the needed features.

Component: #4:The purpose for the anomaly finder is to check/verify the log sequences it receives and to flag any type of outliers. The input that this finder would receive will be coming from the feature extractor from the previous feature.

Component #5:The alert system is in place to alert the user to any anomalies that the anomaly finder detects. Users will be alerted to any finds, or if there were no actual anomalies in the features that it checked. This could be considered the last line of defense in the system to detect the anomalies.

Constraints

- Accessing data from users introduces an extra level of security needed to protect said data.
- Storing user data comes with constraints on storage depending on which method is used or if user logs will be permanently stored.
- More users accessing the system can add overhead leading to a decrease in performance.
- Making sure the logs that are first introduced to the “log gatherer” do not contain any type of malicious code
- Ensure the concepts of the C.I.A Triad are being upheld through the whole process going from component to component

Resources

- Deadlocks and race conditions are possible as multiple systems can try to parse logs to the system at once. To address these, message queues could be deployed.

Development

Log Gatherer

- Needs to gather the LogPai/LogHub data
- Needs Zondo ID to download dataset from hosted site
- Downloads and caches logs locally
- Supports two methods: reading logs from memory and streaming logs (both have tradeoffs)

Live Log Gatherer

- Logs need to be in original format
- The program grabs calls made by a user to take advantage of Linux calls such as “tail” and etc.
- Calls need to be protected from both from viewing and tampering
- Needs to be streamed to parser, then the feature extractor, then the ML model

Log Parser

- Logs need to be clean and organized
- Separate logs out by timestamp, hostname, process, and message
- Handles both single log parsing and batch processing
- Returns structured dictionaries that are easy to work with
- Could also help alerts be more clear and specific to user

Feature Extractor

- Need list of keywords from system that could indicate anomaly
- Feature list to include 61 different keywords for network, system information, general errors, ect
- Extracts the message length and word count of the log as numerical values
- All of this together will result in a vector with 63 features

Machine Learning Model

- We want the model to use the isolated forest method to determine anomalies
- Train model using dataset then load it to repository
- Model will use all of the numerical values from the feature extractor

Alert System

- Program should receive a value based on the machine learning model's evaluation
- Conditions to be set will trigger the alert system. It could possibly be "What value could indicate an anomaly?"
- Use parser to create clean alerts so that the security admins can be aware of what is suspicious. Why am I getting this alert?

Database Connection - Not applicable

Project Setup

1. First, the log gatherer needs to be crafted, this is the base of our whole project

2. We can then worry about how the model will read each log, but only after we master extracting logs
3. Learn how to split information such as time logged, user, permission
    a. The feature extractor should be created then to help determine what is suspicious, as far as system messages are concerned
    b. Training the model is essential to our project
        i. Training will be done using the loghub dataset found on Github
        ii. The files to load the results are to be included in the repository
        iii. A proper condition to separate anomalies will be determined
    c. The alert system can then be developed
        i. Prior anomaly condition will be used to trigger alert
        ii. If value = anomaly condition, send alert
        iii. Alert goes through Discord notification

Test Plan and Report

Test Objectives

- Ensure model is detecting anomalies
- Logs are being read from the journal
- Ensure alert system is being called when anomalies occur
    - Alerts are specific to the anomaly that occurs
- Model needs to be trained on first run of program
    - Model should load data if trained prior

Scope of Testing

In-scope

- Validity of alerts and model
- Collection of logs
- Training of the model

Out-of-scope

- Overall performance of application
- Performance of concurrent usage

Test Cases

1. Model needs to be able to detect anomalies, so add a print statement to the gatherer to validate that it works
2. Gatherer should be reading logs from the sytstemd journal, so compare parser to the actual system's journal using the journalctl command
3. Alert system needs to be called when an anomaly occurs. In addition, the alerts need to be specific. This can be tested by running the machine and passing logger commands to simulate anomalies. If the alert is sent to the channel and it is clear from the message alone what is happening, then it works.
4. User needs to have a trained model, either by training it upon first run of program or by validating the proper loaded file. This can be tested through print statements and personal validation of files in system

Test Procedures

Steps:

1. Boot up Ubuntu VM
2. Check project folder for trained model file (preferably in an environment without it)
3. Open terminal and run the live log gatherer
    a. If no model file, then check for both the terminal for alert and folder for the file after first run
    b. If there is the model file, then nothing should happen
4. Pass logger commands and check Discord channel for alerts, ensure alerts are accurate to what is being tested
5. Stop running log gatherer, then use the "journalctl" command in terminal to validate what has been passed through

Test Environment

- Ubuntu virtual machine

Test Data

- List of logger commands
- Loghub dataset

Test Report

Test case	Pass	Fail	Severity
Detect Anomalies	X		
Reading from systemd	X		
Alert system works	X		
Alerts are informational	X		
Model is trained	X		
Model is loaded if trained prior	X		

Summary

Our team has developed a working anomaly detection system in a Linux environment. The system's components are able to collaborate with each other to read and analyze logs and send alerts to a Discord server. Comprehensive tests show that the machine is in a basic state of anomaly detection, and with minor tweaks and additions, can be completed with a user-friendly environment and finely tuned performance.

Appendix

References

1. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation Forest. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE. [PDF]
2. Zhu, J., He, S., He, P., Liu, J., & Lyu, M. R. (2023). Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In IEEE International Symposium on Software Reliability Engineering (ISSRE). [arXiv]