

Bio Touch Pass: Handwritten Passwords for Touch screen Biometrics

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

Nali Shiva Shankar (37110485)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC

Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI

CHENNAI - 600 119

March - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with “A” grade by NAAC I 12B Status by UGC
Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of Nali Shiva Shankar (37110485) who have done the Project work as a team who carried out the project entitled “Bio Touch Pass: Handwritten Passwords for Touch screen Biometrics” under my supervision from November 2020 to March 2021.

Mrs. A.C. Santha Sheela M.E.

Internal Guide

Head of the Department

Dr. L. Lakshmanan M.E., Ph.D.,

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

DECLARATION

I Vivethan N (37110846) & Pani Frane (37110540) hereby declare that the Project Report entitled “Bio Touch Pass: Handwritten Passwords for Touch screen Biometrics” done by me under the guidance of Mrs. Santha Sheela at Sathyabama Institute of Science & Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board **of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D .**, Dean, School of Computing and **Dr.L.Lakshmanan M.E., Ph.D.,and Dr.S.Vigneshwari M.E., Ph.D.**, Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. A. Pravin,M.E.,Ph.d., for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

- In cloud computing, containers are operating-system level virtualization abstractions for running isolated systems on a host using a single kernel.
- The use of containers in cloud computing has been steadily increasing. With the emergence of Kubernetes, the management of applications inside containers (or pods) is simplified.
- Kubernetes allows automated actions like self-healing, scaling, rolling back, and updates for the application management.
- During application deployment and execution in the pod, a cryptomining process, started by a hidden malware executable can be run in the background, and a method to detect malicious cryptomining software running inside Kubernetes pods is needed.
- One feasible strategy is to use machine learning (ML) to identify and classify pods based on whether or not they contain a running process of cryptomining.
- In addition to such detection, the system administrator will need an explanation as to the reason(s) of the ML's classification outcome.
- In this article, we describe the design and implementation of an ML-based detection system of anomalous pods in a Kubernetes cluster by monitoring Linux-kernel system calls (syscalls).
- Several types of cryptominers images are used as containers within an anomalous pod, and several ML models are built to detect such pods in the presence of numerous healthy cloud workloads.

TABLE OF CONTENTS

ABSTRACT

V

List of Figures

VII

CHAPTER No.	TITLE	PAGE No.
1.	INTRODUCTION	8
	1.1 CRYPTOMINING	8
	1.2 CRYPTOMINING SIGNATURES	10
	1.3 MACHINE LEARNING	11
	1.4 MOTIVATION	14
2.	LITERATURE SURVEY	16
3.	SYSTEM ANALYSIS	23
	3.1 EXISISTING SYSTEM	23
	3.2 PROPOSED SYSTEM	23
	3.3 ARCHITECTURE DIAGRAM	26
	3.4 MODULES	26
4.	SOFTWARE ENVIRONMENT	28
5.	RESULTS	53
	SCREENSHOTS	53
6.	CONCLUSION AND FUTURE WORK	59

	REFERENCES	60
--	-------------------	-----------

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
3.1	SYSTEM ARCHITECTURE	26
5.1	UPLOAD FILE	53
5.2	NODE INITIALIZATION	54
5.3	USING NORMAL CASE	55
5.4	USING CRYPTOMINING CASE	56
5.5	SEND FAKE DATA	57
5.6	DATA INTEGRATION	57
5.7	ATTACK DETECTION	58
5.8	VIEW IP ADDRESS	58

CHAPTER 1

INTRODUCTION

1.1 CRYPTOMINING

The term crypto mining means gaining cryptocurrencies by solving cryptographic equations through the use of computers. This process involves validating data blocks and adding transaction records to a public record (ledger) known as a blockchain. In a more technical sense, cryptocurrency mining is a transactional process that involves the use of computers and cryptographic processes to solve complex functions and record data to a blockchain. In fact, there are entire networks of devices that are involved in cryptomining and that keep shared records via those blockchains.

With cryptocurrencies, there's no central authority, nor is there a centralized ledger. That's because cryptocurrencies operate in a decentralized system with a distributed ledger (more on this shortly) known as blockchain. Unlike the traditional banking system, anybody can be directly connected to and participate in the cryptocurrency "system." You can send and receive payments without going through a central bank. That's why it's called decentralized digital currency. But in addition to being decentralized, cryptocurrency is also a distributed system. This means the record (ledger) of all transactions is publicly available and stored on lots of different computers. This differs from the traditional banks we mentioned earlier, which are centralized systems. But without a central bank, how are transactions verified before being added to the ledger? Instead of using a central banking system to verify transactions (for example, making sure the sender has enough money to make the payment), cryptocurrency uses cryptographic algorithms to verify transactions. And that's where bitcoin miners come in. Performing the cryptographic calculations for each transaction adds up to a lot of

computing work. Miners use their computers to perform the cryptographic work required to add new transactions to the ledger.

In a nutshell, crypto miners verify the legitimacy of transactions in order to reap the rewards of their work in the form of cryptocurrencies. To understand how most cryptocurrency mining works in a more technical sense, you first need to understand the technologies and processes behind it. This includes understanding what blockchain is and how it works.

The first thing to know is that two things are central to the concept of blockchain: public key encryption and math. While I'm definitely a fan of the first, I'll admit that the latter isn't my strong suit. However, public key cryptography (aka public key encryption or asymmetric encryption) and math go together in blockchains like burgers and beer.

Traditional cryptocurrencies such as Bitcoin use a decentralized ledger known as blockchain. A blockchain is a series of chained data blocks that contain key pieces of data, including cryptographic hashes. These blocks, which are integral to a blockchain, are groups of data transactions that get added to the end of the ledger. Not only does this add a layer of transparency, but it also serves as an ego inflator when people get to see their transactions being added (chained) to the blockchain. Even though it doesn't have their names listed on it, it often still evokes a sense of pride and excitement.

1.2 CRYPTOMINING SIGNATURES

Cryptocurrency mining malware refers to software developed to use the computer's resources for cryptocurrency mining without a user's explicit permission. Attackers have attempted to profit from cryptocurrency mining by harnessing the processing power of a large numbers of computers, smartphones and other electronic devices. The detection of cryptocurrency malware has been performed by generating its signatures in terms of power consumption, network traffic behavior, operating system processes, and patterns in hardware performance counters. In, an anatomy of the browser-based cryptomining is presented, in which the attacker infects a web page with Java- Script code that auto-executes when the web page is loaded by the victim's browser. The attacker takes advantage of the browser to activate the necessary JavaScript mining module. The term "illegal leverage" states that javascript is used maximally and forcibly taking full privilege without the victim's consent for a mining operation. The unauthorized execution of JavaScript can therefore be used as a signature for cryptomining malware in this scenario. If the browser behavior is measured using any profiling metrics (e.g., syscalls or processor/ memory metrics like instruction per clock cycle, CPU utilization, virtual memory page faults, context switches, etc.) a definite pattern of those metrics is marked for the legitimate and healthy operation. If the browser is hijacked by the mining operation, a significant deviation is shown by such profiling metric.

Such a deviation is called a mining signature of the browser. In the Windows operating system, mining is run as an executable file in memory that establishes an alteration in the system registry. In this scenario, the monitoring of registry can signal the presence of malware. Network signature extraction is also possible because mining programs contact the central botnet server to register its presence and to download

relevant files depending on the architecture of the victim's system. The network transactions generate significant network traffic before the actual cryptomining begins. Tracing such traffic is relatively easy because the communication is unencrypted. The cryptominer signature is extracted in terms of power consumption for IoT devices.

1.3 MACHINE LEARNING

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs. Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine

learning methodologies, or analyze the impact of machine learning processes. In this tutorial, we'll look into the common machine learning methods of supervised and unsupervised learning, and common algorithmic approaches in machine learning, including the k-nearest neighbor algorithm, decision tree learning, and deep learning. We'll explore which programming languages are most used in machine learning, providing you with some of the positive and negative attributes of each. Additionally, we'll discuss biases that are perpetuated by machine learning algorithms, and consider what can be kept in mind to prevent these biases when building algorithms.

1.3.1 Machine Learning Methods

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. Two of the most widely adopted machine learning methods are supervised learning which trains algorithms based on example input and output data that is labeled by humans, and unsupervised learning which provides the algorithm with no labeled data in order to allow it to find structure within its input data.

1.3.1.1 Supervised Learning

In supervised learning, the computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to “learn” by comparing its actual output with the “taught” outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabeled data.

For example, with supervised learning, an algorithm may be fed data with images of sharks labeled as fish and images of oceans labeled as water. By being trained on this data, the supervised learning algorithm should be able to later identify unlabeled shark images as fish and unlabeled ocean images as water. A common use case of supervised learning is to use historical data to predict statistically likely future events. It may use historical stock market information to anticipate upcoming fluctuations, or be employed to filter out spam emails. In supervised learning, tagged photos of dogs can be used as input data to classify untagged photos of dogs.

1.3.1.2 Unsupervised Learning

In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable. The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases. With this data fed into an unsupervised learning algorithm, it may be determined that women of a certain age range who buy unscented soaps are likely to be pregnant, and therefore a marketing campaign related to pregnancy and baby products can be targeted to this audience in order to

increase their number of purchases. Without being told a “correct” answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next. In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together.

1.4 MOTIVATION OF THE PROJECT

In public cloud computing services, access to the hardware resources is typically not available to the customer. Instead, the Linux-kernel system calls at the operating system level can be used as a proxy to signal the possibility of threat in a running container. The system call (syscall) is the fundamental interface between an application and the Linux kernel. A syscall is generated every time the application interacts with the Linux-kernel. Cryptominers have to repeatedly run a core Proof-of-Work (PoW) algorithm that the currency is based on. Such repeated runs would result in the repeated occurrence of particular patterns for certain syscalls. System call monitoring helps to track such patterns, and any unanticipated change in the patterns of an application can signal the presence of a threat in the container. Under this scenario, an unusual syscall pattern can be used as an alert for cryptomining. Prior research that uses syscalls and behavioral models as detection mechanism while research using neural network models.

Most of the anomaly detection models are considered as black-boxes where no information is returned to the user regarding the cause of the anomaly classification. Yet, this information must be transparent to the

system administrator who will need an explainable classification model in order to take the appropriate action. An explainable model generates an auditable set of explanations that describe key factors associated with the prediction. It can recommend the critical signals that need to be carefully monitored or recommend specific actions such as increasing the sampling frequency to get finer-grain details of the event that is responsible for the anomaly. It can also explain the association among the signals which are needed to manage the false prediction rate. The best use of such association rules is in fault tracing where impact at one pod might be due to some event in another pod, which cascades in some manner to a third pod, and so on. Explanations obtained from machine learning models help trace key features or sequences and eventually detect the root cause. In this work, a methodology is formulated and implemented to detect cryptominer anomalies using system calls as proxies for mining events and using an explainable machine learning (ML) model as the cryptomining detector.

CHAPTER 2

LITERATURE SURVEY

2.1 Crypto mining attacks in information systems: An emerging threat to cyber security

AUTHORS: A. Zimba, Z. Wang, M. Mulenga

The popularity of cryptocurrencies has continued to grow drastically over the past decade and this has drawn significant attention to various threat actors. Cybercriminals are now employing unconventional means to acquire cryptocurrencies at the expense of benign Internet users. This paper investigates the state-of-the-art crypto mining attacks by examining the malware code and the behavioral analysis upon execution. It examines the two most common attack approaches; web browser-based crypto mining which leverages JavaScript and installable binary crypto mining where the malware runs in memory. Furthermore, the paper investigates how cybercriminals endeavor to establish a persistence mechanism and avoid detection. The results from static and dynamic analysis uncover the techniques employed by the malware to exploit potential victims. Indicators of compromise are drawn from the uncovered artifacts which can be used as inputs to intrusion detection systems to help mitigate such cyber-attacks.

2.2) Evaluation of deep learning methods efficiency for malicious and benign system calls classification on the AWSCTD

AUTHORS: D. Ceponis and N. Goranin

The increasing amount of malware and cyberattacks on a host level increases the need for a reliable anomaly-based host IDS (HIDS) that would be able to deal with zero-day attacks and would ensure low false

alarm rate (FAR), which is critical for the detection of such activity. Deep learning methods such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are considered to be highly suitable for solving data-driven security solutions. Therefore, it is necessary to perform the comparative analysis of such methods in order to evaluate their efficiency in attack classification as well as their ability to distinguish malicious and benign activity. In this article, we present the results achieved with the AWSCTD (attack-caused Windows OS system calls traces dataset), which can be considered as the most exhaustive set of host-level anomalies at the moment, including 112.56 million system calls from 12110 executable malware samples and 3145 benign software samples with 16.3 million system calls. The best results were obtained with CNNs with up to 90.0% accuracy for family classification and 95.0% accuracy for malicious/benign determination. RNNs demonstrated slightly inferior results. Furthermore, CNN tuning via an increase in the number of layers should make them practically applicable for host-level anomaly detection.

2.3) Android malware detection based on system call sequences and LSTM

AUTHORS: X. Xiao, S. Zhang, F. Mercaldo, G. Hu

As Android-based mobile devices become increasingly popular, malware detection on Android is very crucial nowadays. In this paper, a novel detection method based on deep learning is proposed to distinguish malware from trusted applications. Considering there is some semantic information in system call sequences as the natural language, we treat one system call sequence as a sentence in the language and construct a classifier based on the Long Short-Term Memory (LSTM) language model. In the classifier, at first two LSTM models are trained respectively

by the system call sequences from malware and those from benign applications. Then according to these models, two similarity scores are computed. Finally, the classifier determines whether the application under analysis is malicious or trusted by the greater score. Thorough experiments show that our approach can achieve high efficiency and reach high recall of 96.6% with low false positive rate of 9.3%, which is better than the other methods.

2.4) Exploring adversarial examples in malware detection

AUTHORS: O. Suciu, S. Coull, and J. Johns

The convolutional neural network (CNN) architecture is increasingly being applied to new domains, such as malware detection, where it is able to learn malicious behavior from raw bytes extracted from executables. These architectures reach impressive performance with no feature engineering effort involved, but their robustness against active attackers is yet to be understood. Such malware detectors could face a new attack vector in the form of adversarial interference with the classification model. Existing evasion attacks intended to cause misclassification on test-time instances, which have been extensively studied for image classifiers, are not applicable because of the input semantics that prevents arbitrary changes to the binaries. This paper explores the area of adversarial examples for malware detection. By training an existing model on a production-scale dataset, we show that some previous attacks are less effective than initially reported, while simultaneously highlighting architectural weaknesses that facilitate new attack strategies for malware classification. Finally, we explore how generalizable different attack strategies are, the trade-offs when aiming to increase their effectiveness, and the transferability of single-step attacks.

2.5) Criteria for learning without forgetting in artificial neural networks

AUTHORS: R. Karn, P. Kudva, and I. Elfadel

Task progressive learning without catastrophic forgetting using artificial neural networks (ANNs) has demonstrated viability and promise. Due to the large number of ANN hyper-parameters, a model already trained over a group of tasks can further learn a new task without forgetting the previous tasks. Several algorithms have been proposed for progressive learning, including synaptic weight consolidation, ensemble, rehearsal, and sparse coding. One major problem with such methods is that they fail to detect the congestion in the ANN shared parameter space to indicate the saturation of the existing network and its inability to add new tasks using progressive learning. The detection of such saturation is especially needed to avoid the catastrophic forgetting of old trained task and the concurrent loss in their generalization quality. In this paper, we address such problem and propose a methodology for ANN congestion detection. The methodology is based on computing the Hessian of the ANN loss function at the optimal weights for a group of previously learned tasks. Since the Hessian calculation is compute-intensive, we provide a set of approximation heuristics that are computationally efficient. The algorithms are implemented and analyzed in the context of two cloud network security datasets, namely, UNSW-NB15 and AWID, as well as the MNIST image recognition dataset. Results show that the proposed congestion metrics give an accurate assessment of the ANN progressive learning capacity for these various datasets. Furthermore, the results show that models that have more features exhibit higher congestion thresholds and are therefore more amenable to progressive learning.

2.6) Adversarial deep learning for robust detection of binary encoded malware

AUTHORS: A. Al-Dujaili, A. Huang, E. Hemberg

Malware is constantly adapting in order to avoid detection. Model based malware detectors, such as SVM and neural networks, are vulnerable to so-called adversarial examples which are modest changes to detectable malware that allows the resulting malware to evade detection. Continuous-valued methods that are robust to adversarial examples of images have been developed using saddle-point optimization formulations. We are inspired by them to develop similar methods for the discrete, e.g. binary, domain which characterizes the features of malware. A specific extra challenge of malware is that the adversarial examples must be generated in a way that preserves their malicious functionality. We introduce methods capable of generating functionally preserved adversarial malware examples in the binary domain. Using the saddle-point formulation, we incorporate the adversarial examples into the training of models that are robust to them. We evaluate the effectiveness of the methods and others in the literature on a set of Portable Execution~(PE) files. Comparison prompts our introduction of an online measure computed during training to assess general expectation of robustness.

2.8) Feedback autonomic provisioning for guaranteeing performance in MapReduce systems

AUTHORS: M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu

Companies have a fast growing amounts of data to process and store, a data explosion is happening next to us. Currently one of the most common approaches to treat these vast data quantities are based on the MapReduce parallel programming paradigm. While its use is widespread in the industry, ensuring performance constraints, while at the same time minimizing costs, still provides considerable challenges. We propose a coarse grained control theoretical approach, based on techniques that

have already proved their usefulness in the control community. We introduce the first algorithm to create dynamic models for Big Data MapReduce systems, running a concurrent workload. Furthermore, we identify two important control use cases: relaxed performance-minimal resource and strict performance. For the first case we develop two feedback control mechanism. A classical feedback controller and an even-based feedback, that minimises the number of cluster reconfigurations as well. Moreover, to address strict performance requirements a feedforward predictive controller that efficiently suppresses the effects of large workload size variations is developed. All the controllers are validated online in a benchmark running in a real 60 node MapReduce cluster, using a data intensive Business Intelligence workload. Our experiments demonstrate the success of the control strategies employed in assuring service time constraints.

2.9) Spatio-temporal convolutional sparse auto-encoder for sequence classification

AUTHORS: M. Baccouche, F. Mamalet, C. Wolf

We present in this paper a novel learning-based approach for video sequence classification. Contrary to the dominant methodology, which relies on hand-crafted features that are manually engineered to be optimal for a specific task, our neural model automatically learns a sparse shift-invariant representation of the local 2D+t salient information, without any use of prior knowledge. To that aim, a spatio-temporal convolutional sparse autoencoder is trained to project a given input in a feature space, and to reconstruct it from its projection coordinates. Learning is performed in an unsupervised manner by minimizing a global parametrized objective function. The sparsity is ensured by adding a sparsifying logistic between the encoder and the decoder, while the shift-invariance is handled by including an additional hidden variable to the objective function. The

temporal evolution of the obtained sparse features is learned by a long short-term memory recurrent neural network trained to classify each sequence. We show that, since the feature learning process is problem-independent, the model achieves outstanding performances when applied to two different problems, namely human action and facial expression recognition. Obtained results are superior to the state of the art on the GEMEP-FERA dataset and among the very best on the KTH dataset.

2.10) Cryptomining application fingerprinting method

AUTHORS: D. Draghicescu, A. Caranica, A. Vulpe, and O. Fratu

Computing power has increased exponentially in the last decades, and more and more consumer devices are now permanently connected to the internet, to form the so called “Internet of Things” (IoT). Following this trend, security threats and vulnerabilities have increased drastically over a short period of time, as more and more devices are inter-connected and not properly secured. Lately, the growing popularity of Bitcoin and other cryptocurrencies are generating a lot of hype and concern, among security specialists, as unwanted mining applications on an end-user device can potentially pose both security risks and increased operating costs. There are multiple ways of detecting unwanted applications at the user-level, by means of URL blocking and application fingerprinting, and this paper will cover a detection method proposed by our group.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Several methods to analyze syscall patterns are available. A method that provides the best accuracy across cryptominers needs to be identified along with the best possible explanation for such identification.

One such method is where a histogram of syscalls is created to find the distribution of distinct syscalls in each time window. Another method is given where the semantics of each syscall is interpreted to detect infected pods. A flow-graph analysis is used in [20] to deduce the relationships between different syscalls and generate application signatures. A Markov chain is mostly used to graph the syscalls as a sequence of events in which the probability of each event depends only on the state attained in the previous event. Similarly a weighted directed graph is built using syscalls for Android malware detection. Such graph is used as a malware signature and is compared with other container syscalls graphs to detect the anomaly.

3.1.1 Disadvantages of existing system

- ✓ They require a significant amount of manual intervention to analyze and interpret syscall semantics.
- ✓ They are difficult to use in the analysis of applications that produce a large number of distinct syscalls in a small time window.

3.2 PROPOSED SYSTEM

Cryptominers prevent the deployed application from using full container resources. Before deploying, booting or running the desired application, it is therefore crucial to perform the health checks on the container base image. In this work, we design an ML-based cryptomining container detection framework using syscalls as a monitoring mechanism. The cryptomining anomaly detection is based on the principle of establishing an application behavior baseline and then evaluating subsequent events against this baseline. Anything “too far” from this baseline can be regarded as anomalous and should be investigated. We use several statistical and rule-based ML algorithms, and back up their

detection results with several explainability tools to investigate the cause of the ML outcomes. These ML algorithms are then compared in terms of their performance metrics.

A methodology for anomaly detection through system calls in the Kubernetes pods is proposed, designed, and implemented. Several types of cryptominer images are used in the creation of anomalous pods. Proxies based on Linuxkernel syscalls are extracted and compared against healthy applications that exhibit similar domain behavior as the cryptominers. Four different ML algorithms are used for classifying a given pod as either a crypto-hijacked or a normal pod. These algorithms are compared in terms of accuracy, runtime, and resource utilization.

This paper makes the following specific contributions:

- 1) Design and implementation of a novel automated cryptomining pod detection in a Kubernetes cluster.
- 2) Development and implementation of real-time, syscall extraction methods for Kubernetes pods.
- 3) Implementation of statistical and rule-based ML models to detect anomalous pods.
- 4) Implementation of two statistical explainability mechanisms for ML models: one using open-source components and another with home-grown software.
- 5) Comparative analysis of explainable ML implementations with their differences quantified using welldefined performance metrics

3.2.1 Advantages of proposed system

- ✓ The syscalls frames from such n-grams achieve an aggregate anomaly prediction accuracy of more than 78 percent.
- ✓ Improve performance
- ✓ Less computation time

3.3 SYSTEM ARCHITECTURE

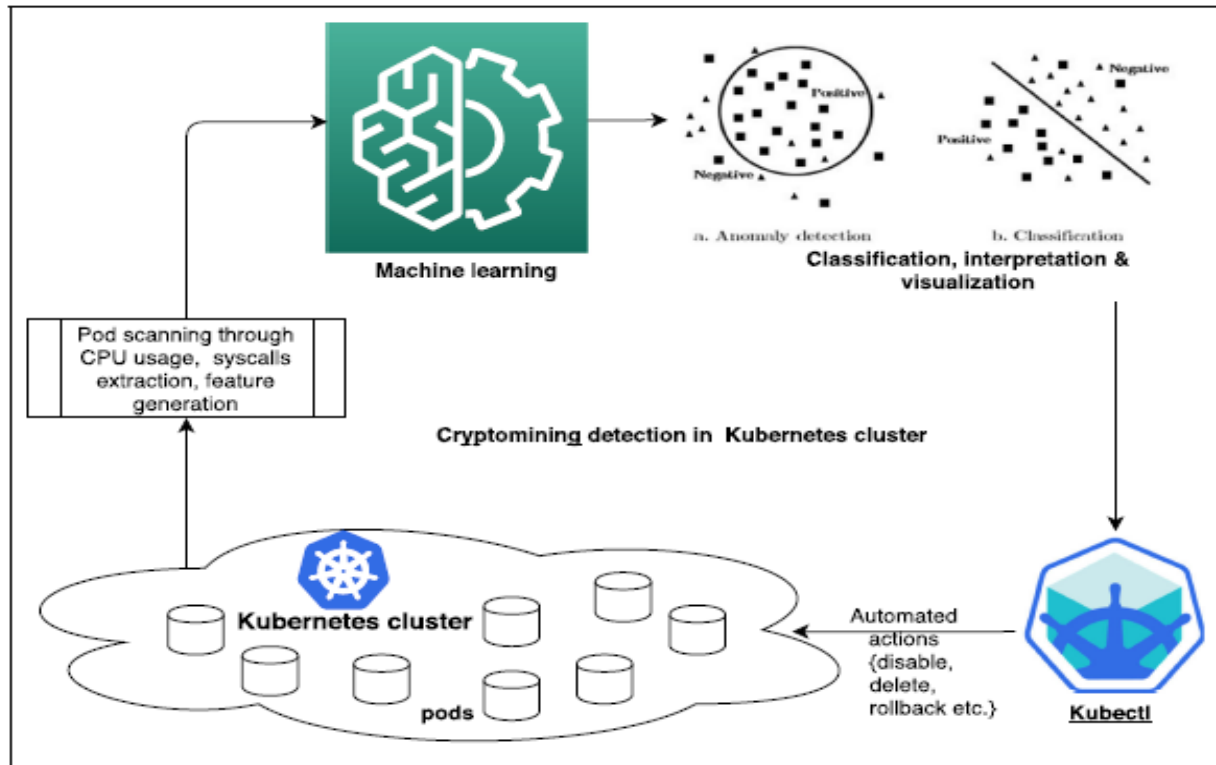


Fig 3.1 System architecture

3.4 MODULES

- Topology construction
- Collection of path backscatter messages
- Cryptomining detection mechanism using machine learning

3.4.1 Topology Construction

The topology is the arrangement of nodes in the simulation area. The routers are connected in mesh topology. In which each routers are connected to each other via other routers (Path). In our simulation, we are using 11 nodes as the router node and 20 nodes as the client-server node. Totally we are having 31 nodes in our network. Each host is connected via routers. Each host has multiple paths to reach a single destination node in the network.

3.4.2 Collection of path backscatter messages

Though path backscatter can happen in any spoofing based attacks, it is not always possible to collect the path backscatter messages, as they are sent to the spoofed addresses. We classify spoofing based attacks into four categories, and discuss whether path backscatter messages can be collected in each category of attacks. Single Source, Multiple Destinations: In such attacks, all the attack packets have the same source IP address.

3.4.3 Cryptomining detection mechanism using machine learning

Cryptomining detection is actually composed by a set of mechanisms. The basic mechanism, which is based on topology and routing information, is illustrated below. Whenever a path backscatter message whose source is router r (named reflector) and the original destination is od is captured, the most direct inference is that the packet from attacker to od should bypass r . We use a machine learning mechanism in spoofing origin tracking.

CHAPTER 4

Software Environment

Java Technology

Java technology is both a programming language and a platform.

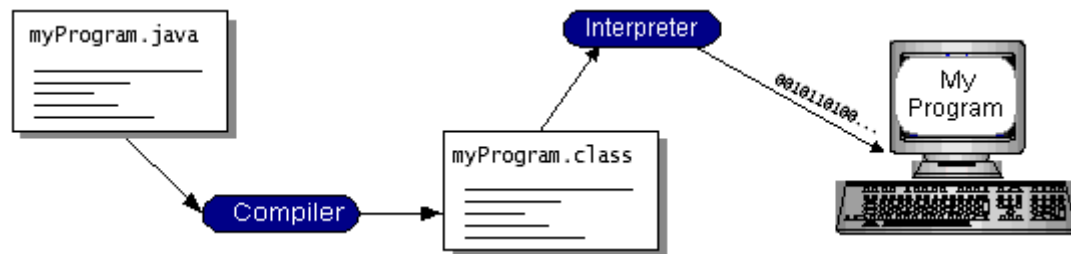
The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

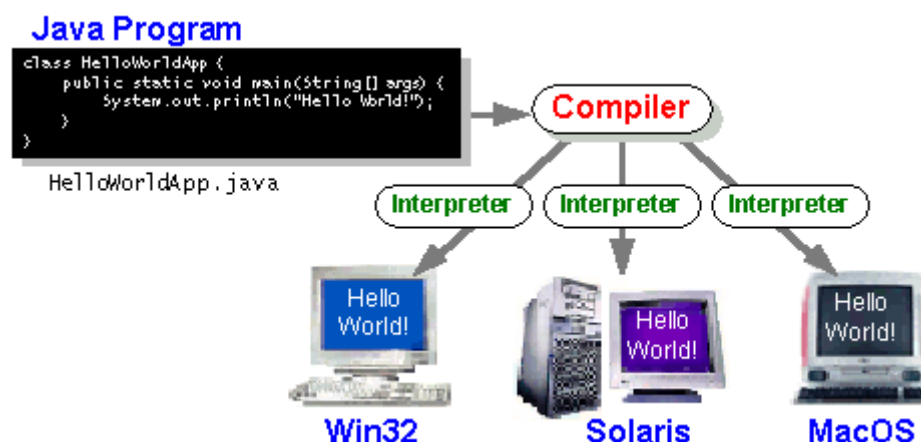
- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation

happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



The Java Platform

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

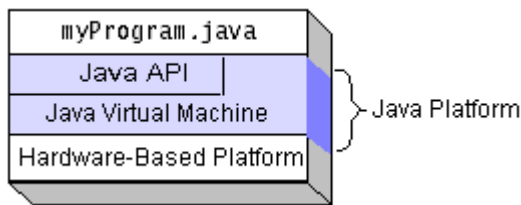
The Java platform has two components:

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, *What Can Java Technology Do?* Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

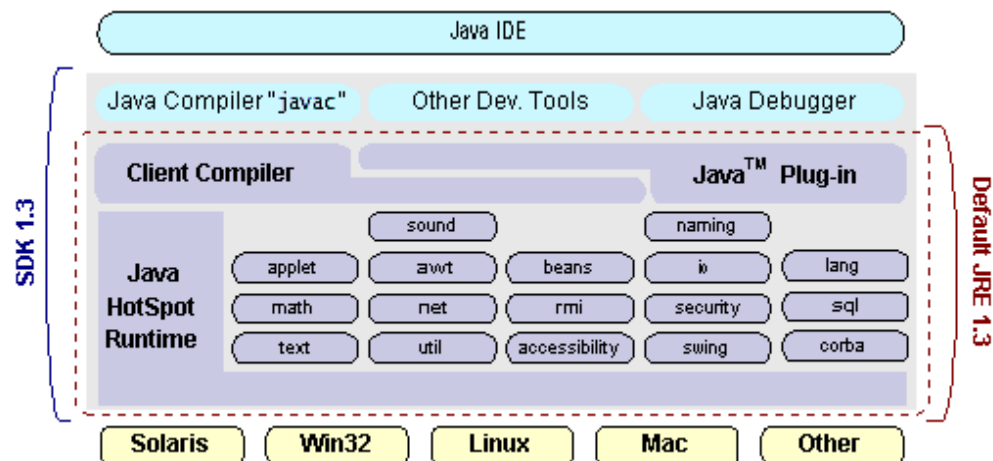
An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing

the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials:** Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets:** The set of conventions used by applets.
- **Networking:** URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization:** Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security:** Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components:** Known as JavaBeans™, can plug into existing component architectures.
- **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC™):** Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



How Will Java Technology Change My Life?

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.

- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
- **Distribute software more easily:** You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

ODBC

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a de facto standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now,

ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel

spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

JDBC

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or drivers. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. SQL Level API

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application

programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

2. SQL Conformance

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

3. JDBC must be implemental on top of common database interfaces

The JDBC SQL API must “sit” on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

4. Provide a Java interface that is consistent with the rest of the Java system

Because of Java’s acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

5. Keep it simple

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

6. Use strong, static typing wherever possible

Strong typing allows for more error checking to be done at compile time; also, less error appear at runtime.

7. **Keep the common cases simple**

Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using Java [Networking](#).

And for dynamically updating the cache table we go for MS [Access](#) database.

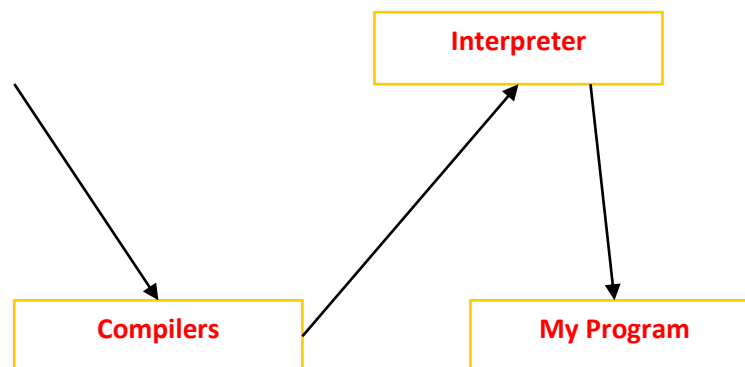
Java ha two things: a programming language and a platform.

Java is a high-level programming language that is all of the following

Simple	Architecture-neutral
Object-oriented	Portable
Distributed	High-performance
Interpreted	multithreaded
Robust	Dynamic
Secure	

Java is also unusual in that each Java program is both compiled and interpreted. With a compile you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.



Java Program

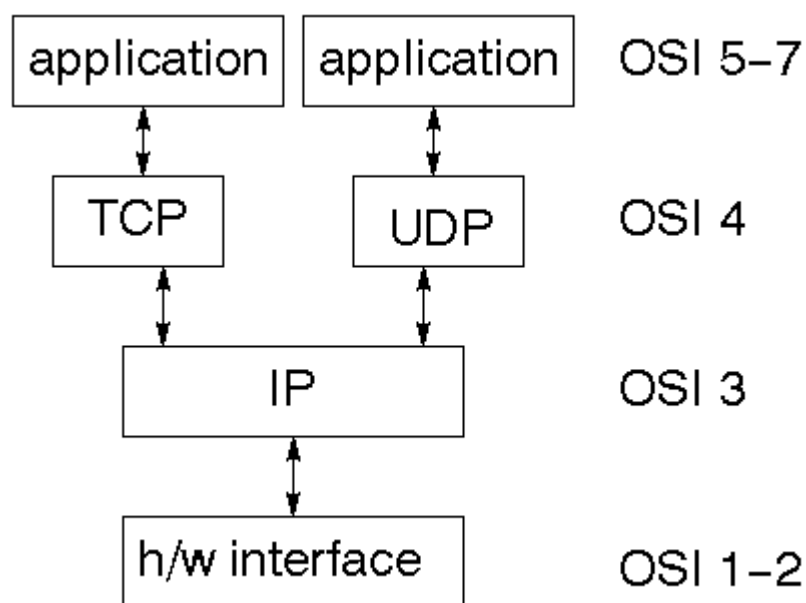
You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make “write once, run anywhere” possible. You can compile your Java program into byte codes on my platform that has a Java compiler. The byte codes can then be run any implementation of the Java VM. For example, the same Java program can run Windows NT, Solaris, and Macintosh.

Networking

TCP/IP stack

The TCP/IP stack i



s shorter

than the OSI one:

TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

IP datagram's

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP

address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

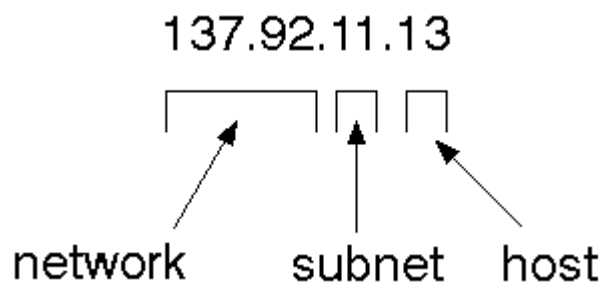
Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

Total address



The 32 bit address is usually written as 4 integers separated by dots.

Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call `socket`. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

Here "family" will be `AF_INET` for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

JFree Chart

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

A consistent and well-documented API, supporting a wide range of chart types;

A flexible design that is easy to extend, and targets both server-side and client-side applications;

Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

JFreeChart is "open source" or, more specifically, [free software](#). It is distributed under the terms of the [GNU Lesser General Public Licence](#) (LGPL), which permits use in proprietary applications.

1. Map Visualizations

Charts showing values that relate to geographical areas. Some examples include: (a) population density in each state of the United States, (b) income per capita for each country in Europe, (c) life expectancy in each country of the world. The tasks in this project include:

Sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas);

Creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XYPlot class in JFreeChart;

Testing, documenting, testing some more, documenting some more.

2. Time Series Chart Interactivity

Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

3. Dashboards

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

4. Property Editors

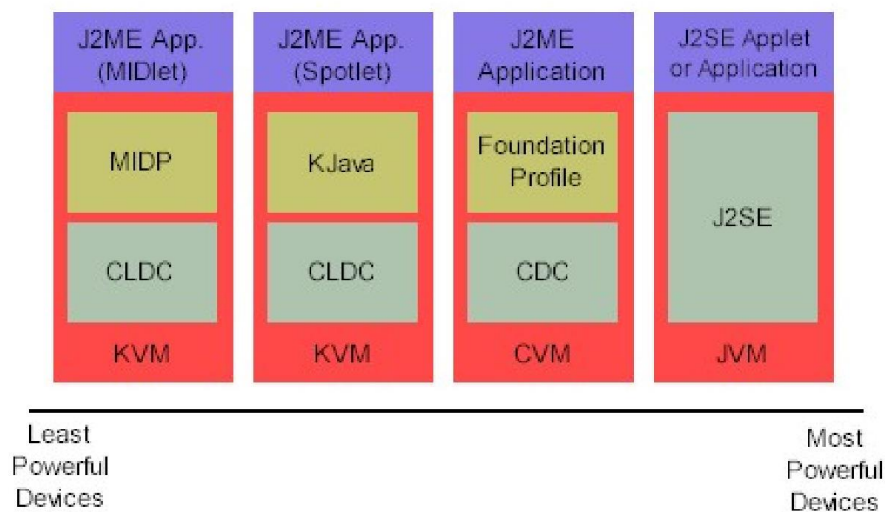
The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or reimplement) this mechanism to provide greater end-user control over the appearance of the charts.

J2ME (Java 2 Micro edition):-

Sun Microsystems defines J2ME as "a highly optimized Java run-time environment targeting a wide range of consumer products, including pagers, cellular phones, screen-phones, digital set-top boxes and car navigation systems." Announced in June 1999 at the JavaOne Developer

Conference, J2ME brings the cross-platform functionality of the Java language to smaller devices, allowing mobile wireless devices to share applications. With J2ME, Sun has adapted the Java platform for consumer products that incorporate or are based on small computing devices.

1. General J2ME architecture



J2ME uses configurations and profiles to customize the Java Runtime Environment (JRE). As a complete JRE, J2ME is comprised of a configuration, which determines the JVM used, and a profile, which defines the application by adding domain-specific classes. The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. We'll discuss configurations in detail in the The profile defines the application; specifically, it adds domain-specific classes to the J2ME configuration to define certain uses for devices. We'll cover profiles in depth in the The following graphic depicts the relationship between the different virtual machines, configurations, and profiles. It also draws a parallel with the

J2SE API and its Java virtual machine. While the J2SE virtual machine is generally referred to as a JVM, the J2ME virtual machines, KVM and CVM, are subsets of JVM. Both KVM and CVM can be thought of as a kind of Java virtual machine -- it's just that they are shrunken versions of the J2SE JVM and are specific to J2ME.

2.Developing J2ME applications

Introduction In this section, we will go over some considerations you need to keep in mind when developing applications for smaller devices. We'll take a look at the way the compiler is invoked when using J2SE to compile J2ME applications. Finally, we'll explore packaging and deployment and the role preverification plays in this process.

3.Design considerations for small devices

Developing applications for small devices requires you to keep certain strategies in mind during the design phase. It is best to strategically design an application for a small device before you begin coding. Correcting the code because you failed to consider all of the "gotchas" before developing the application can be a painful process. Here are some design strategies to consider:

- * Keep it simple. Remove unnecessary features, possibly making those features a separate, secondary application.
- * Smaller is better. This consideration should be a "no brainer" for all developers. Smaller applications use less memory on the device and require shorter installation times. Consider packaging your Java applications as compressed Java Archive (jar) files.

* Minimize run-time memory use. To minimize the amount of memory used at run time, use scalar types in place of object types. Also, do not depend on the garbage collector. You should manage the memory efficiently yourself by setting object references to null when you are finished with them. Another way to reduce run-time memory is to use lazy instantiation, only allocating objects on an as-needed basis. Other ways of reducing overall and peak memory use on small devices are to release resources quickly, reuse objects, and avoid exceptions.

4.Configurations overview

The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. Currently, two configurations exist for J2ME, though others may be defined in the future:

* **Connected Limited Device Configuration (CLDC)** is used specifically with the KVM for 16-bit or 32-bit devices with limited amounts of memory. This is the configuration (and the virtual machine) used for developing small J2ME applications. Its size limitations make CLDC more interesting and challenging (from a development point of view) than CDC. CLDC is also the configuration that we will use for developing our drawing tool application. An example of a small wireless device running small applications is a Palm hand-held computer.

* **Connected Device Configuration (CDC)** is used with the C virtual machine (CVM) and is used for 32-bit architectures requiring more than 2 MB of memory. An example of such a device is a Net TV box.

5.J2ME profiles

What is a J2ME profile?

As we mentioned earlier in this tutorial, a profile defines the type of device supported. The Mobile Information Device Profile (MIDP), for example, defines classes for cellular phones. It adds domain-specific classes to the J2ME configuration to define uses for similar devices. Two profiles have been defined for J2ME and are built upon CLDC: KJava and MIDP. Both KJava and MIDP are associated with CLDC and smaller devices. Profiles are built on top of configurations. Because profiles are specific to the size of the device (amount of memory) on which an application runs, certain profiles are associated with certain configurations.

A skeleton profile upon which you can create your own profile, the Foundation Profile, is available for CDC.

Profile 1: KJava

KJava is Sun's proprietary profile and contains the KJava API. The KJava profile is built on top of the CLDC configuration. The KJava virtual machine, KVM, accepts the same byte codes and class file format as the classic J2SE virtual machine. KJava contains a Sun-specific API that runs on the Palm OS. The KJava API has a great deal in common with the J2SE Abstract Windowing Toolkit (AWT). However, because it is not a standard J2ME package, its main package is `com.sun.kjava`. We'll learn more about the KJava API later in this tutorial when we develop some sample applications.

Profile 2: MIDP

MIDP is geared toward mobile devices such as cellular phones and pagers. The MIDP, like KJava, is built upon CLDC and provides a standard run-time environment that allows new applications and services to be deployed dynamically on end user devices. MIDP is a common,

industry-standard profile for mobile devices that is not dependent on a specific vendor. It is a complete and supported foundation for mobile application

development. MIDP contains the following packages, the first three of which are core CLDC packages, plus three MIDP-specific packages.

- * java.lang

- * java.io

- * java.util

- * javax.microedition.io

- * javax.microedition.lcdui

- * javax.microedition.midlet

- * javax.microedition.rms

CHAPTER 5

SCREENSHOTS

5.1 UPLOAD FILE

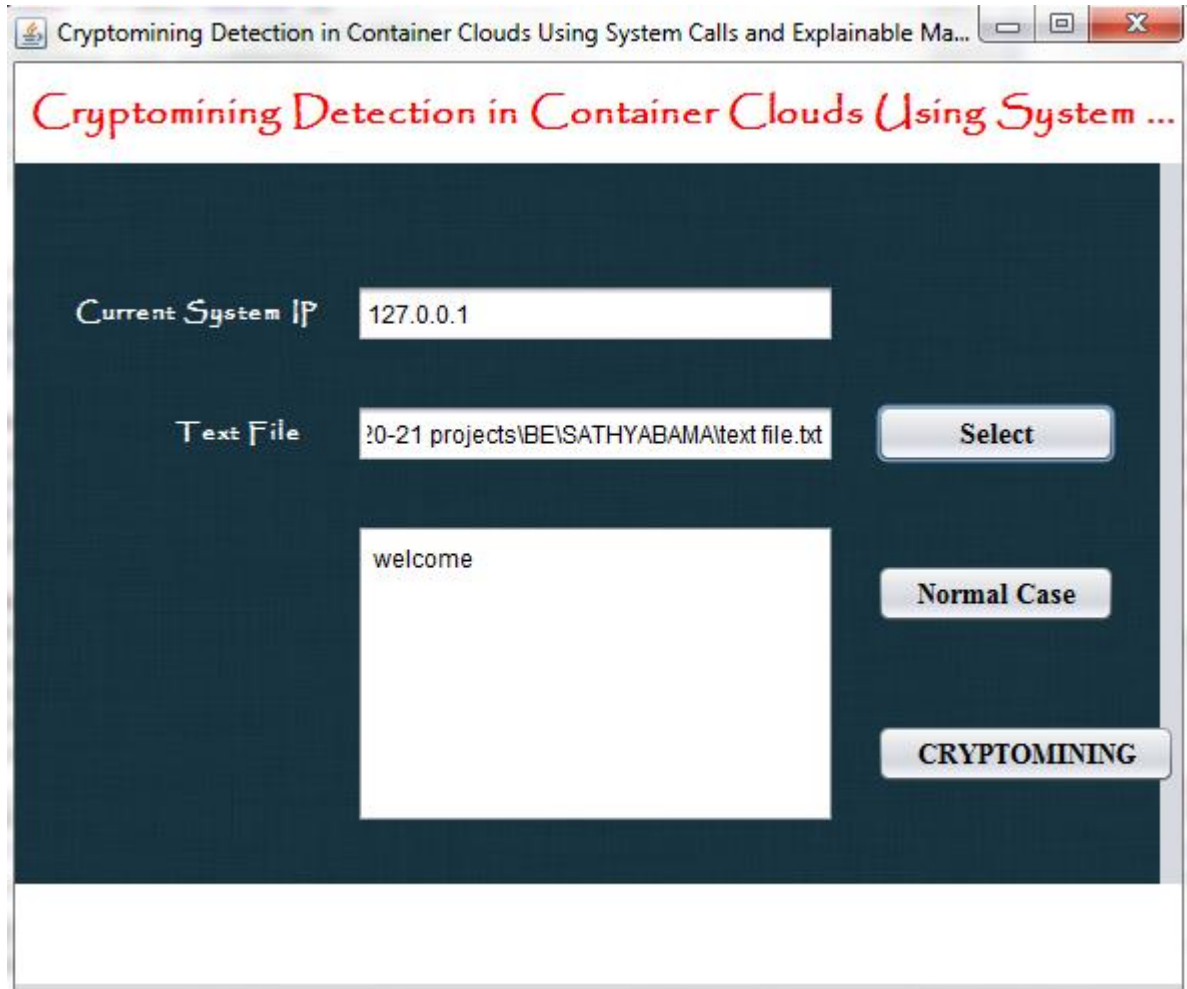


Fig 5.1 Upload file

5.2 NODE INITIALIZATION

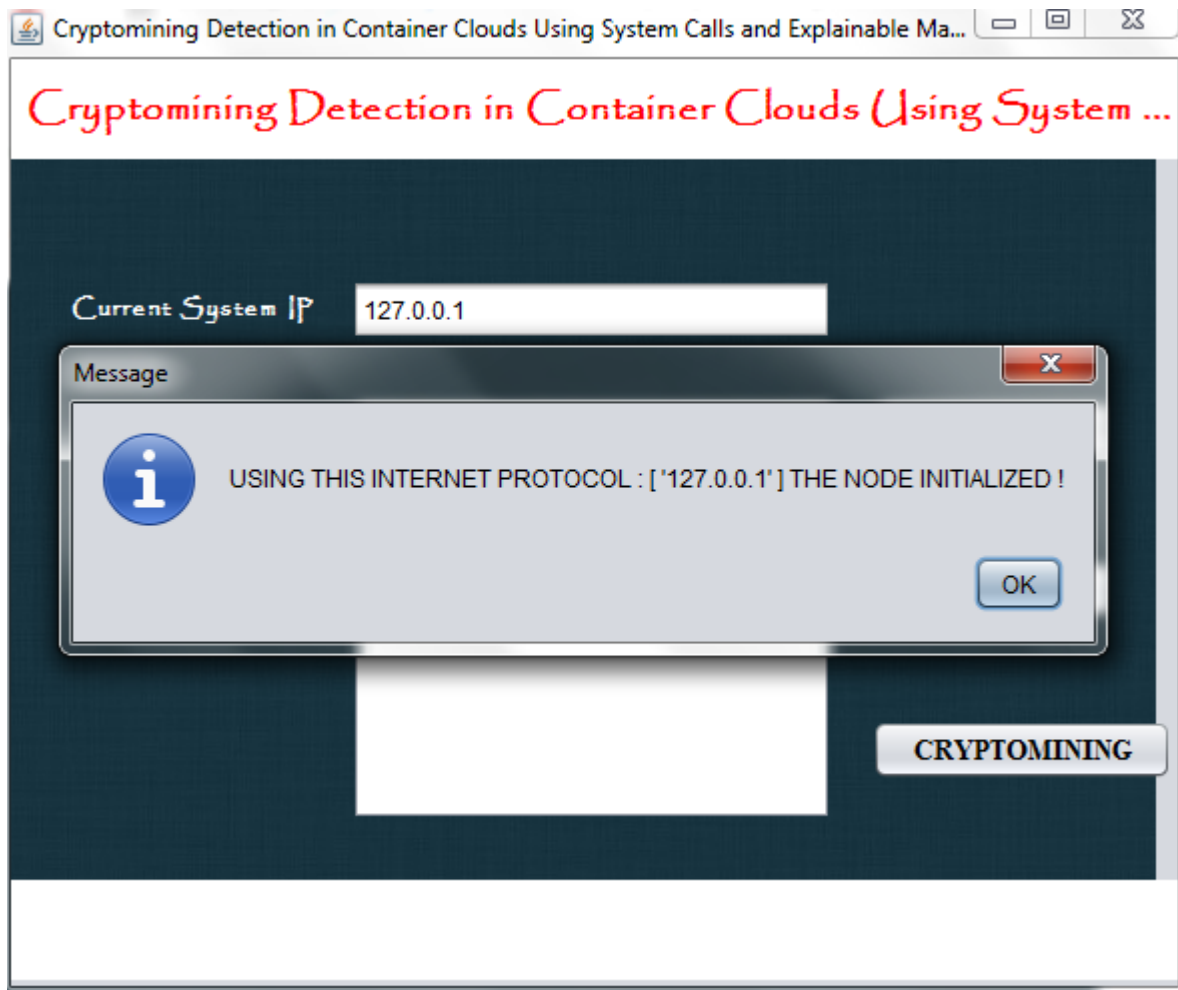


Fig 5.2 Node initialization

5.3 FILE SENDING FROM SOURCE TO DESTINATION USING NORMAL CASE

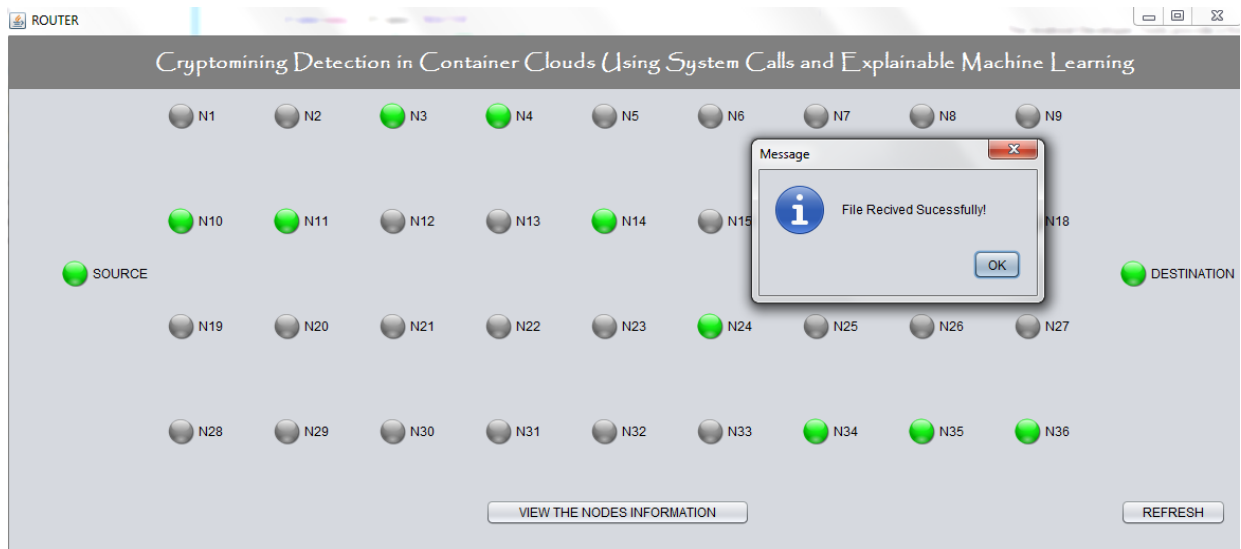


Fig 5.3 File sending from source to destination using normal case

5.4 NODE INITIALIZATION USING CRYPTOMINING CASE

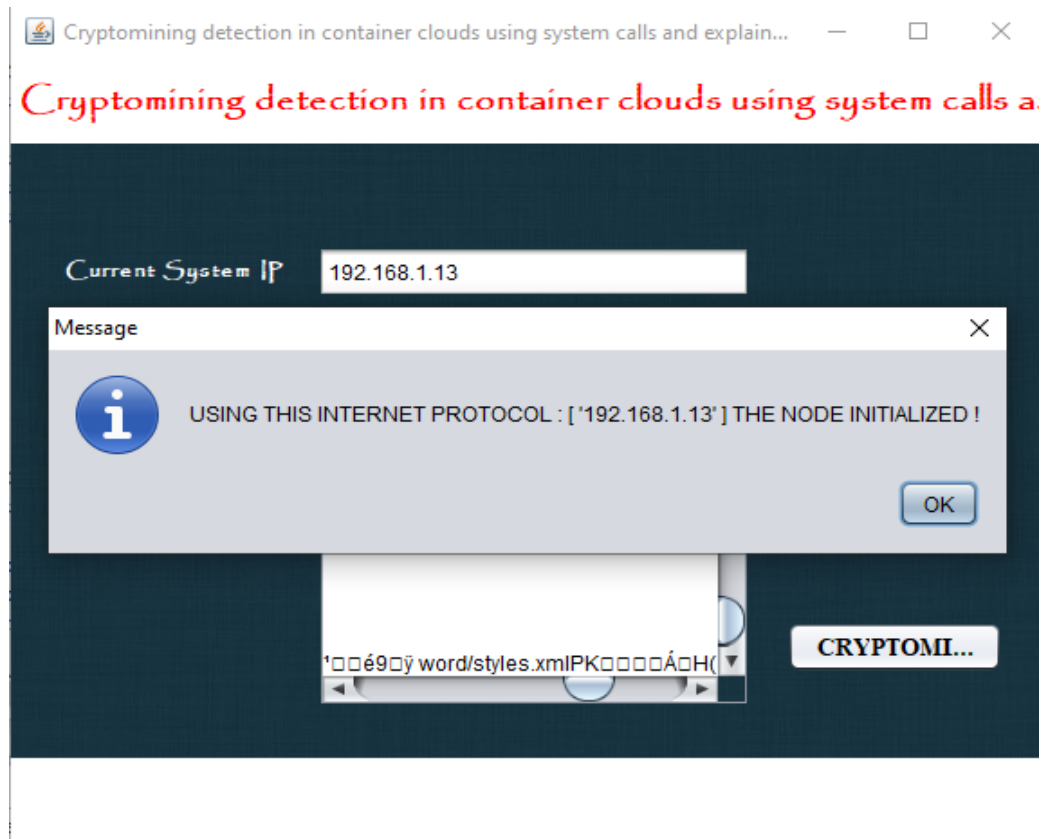


Fig 5.4 Node initialization using cryptomining case

5.5 SEND FAKE DATA

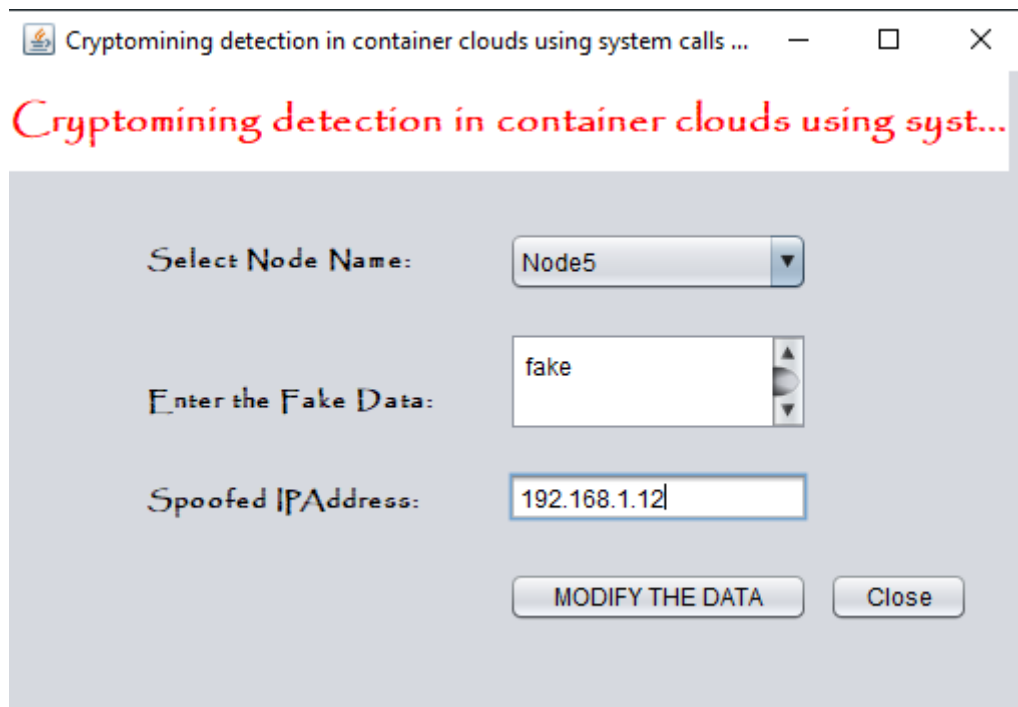


Fig 5.5 Send fake data

5.6 DATA INTEGRATION

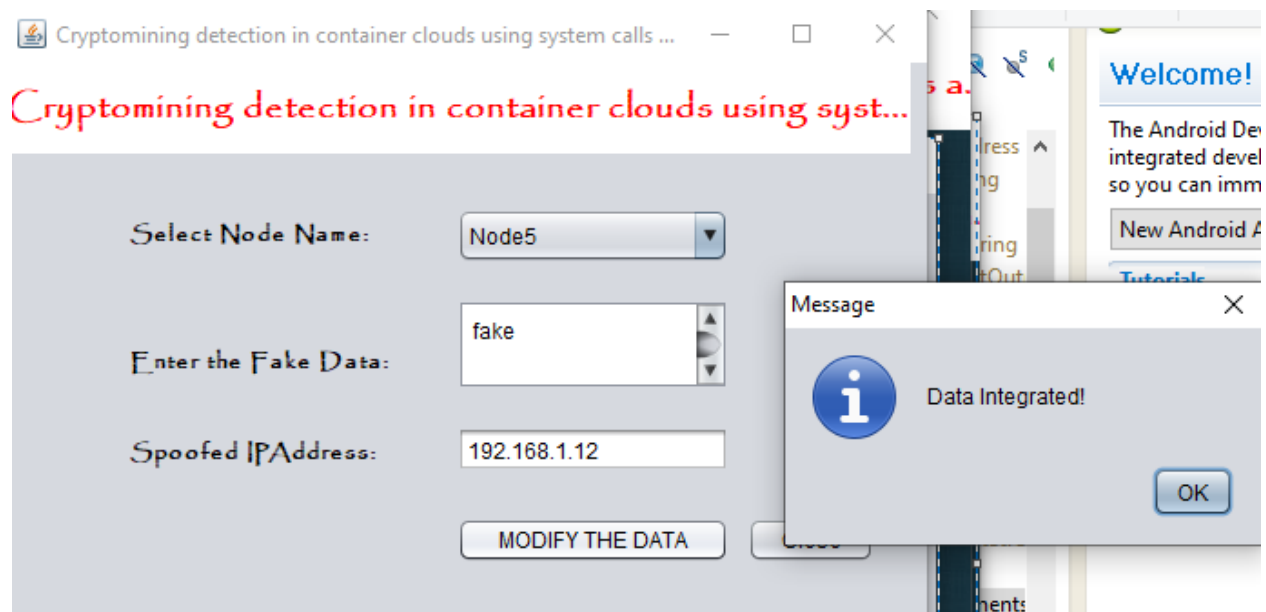


Fig 5.6 Data integration

5.7 ATTACK DETECTION FOR CRYPTOMINING CASE

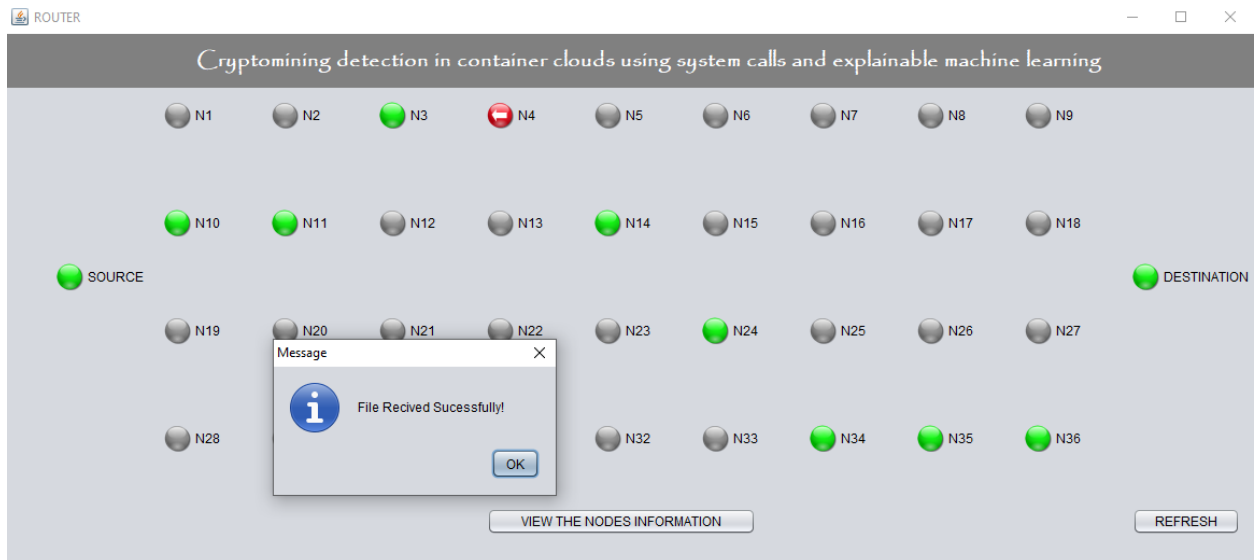


Fig 5.7 Attack detection for cryptomining case

5.8 VIEW ATTACKED IP ADDRESS

ID	NODE NAME	ATTACKER IP	DATA INTEGRITY
1	Node1	null	null
2	Node2	null	null
3	Node3	null	null
4	Node4	192.68.100.11	fake
5	Node5	192.168.1.12	fake
6	Node6	null	null
7	Node7	null	null
8	Node8	null	null
9	Node9	null	null
10	Node10	null	null

Fig 5.8 View attacked IP address

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this paper, an automated pod anomaly detection setup is demonstrated in a Kubernetes cluster to detect cryptomining applications using explainable ML models. The explainability aspect is important for system administrators who must grasp the system-level rationales to support disruptive administrative decisions such as pod removal from a cluster. Several types of cryptomining algorithms may be used to launch an anomalous pod but the patterns of cryptomining system calls have common features that facilitate anomalous pod identification and discrimination against other CPU-intensive applications such as deep-learning, MySQL, Cassandra, Hydoop etc. Three explainable tools and four ML model have been implemented using syscall n-grams as data features. The syscalls frames from such n-grams achieve an aggregate anomaly prediction accuracy of more than 78 percent.

Further, a comparative study of ML explainability among the four models has been performed with the tree decision model found to be the most precise achieving accuracy of more than 97 percent, SHAP and LIME are most efficient while LSTM autoencoder being least amenable to automated explanation extraction because of longer training time and convergence instability.

REFERENCES

- [1] Accessed: Aug. 16, 2018. [Online]. Available: <https://kubernetes.io/>
- [2] A. S. Abed, C. Clancy, and D. S. Levy, "Intrusion detection system for applications using linux containers," in Proc. Int. Workshop Secur. Trust Manage., 2015, pp. 123–135.
- [3] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, and L. Foschini, "Securing the infrastructure and the workloads of linux containers," in Proc. IEEE Conf. Commun. Netw. Secur., 2015, pp. 559–567.
- [4] Accessed: Aug. 16, 2018. [Online]. Available: <https://www.bleepingcomputer.com/news/security/17-backdoored-dockerimages-removed-from-docker-hub/>
- [5] Accessed: Aug. 16, 2018. [Online]. Available: <https://arstechnica.com/information-technology/2018/06/backdoored-images-downloaded5-million-times-finally-removed-from-docker-hub/>
- [6] Accessed: Aug. 16, 2018. [Online]. Available: <https://news.ycombinator.com/item?id=17309883>
- [7] Accessed: Jun. 07, 2020. [Online]. Available: <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf>
- [8] Accessed: Jun. 07, 2020. [Online]. Available: <https://www.guardicore.com/2019/05/nansh0u-campaign-hackers-arsenalgrows-stronger/>
- [9] Accessed: Jun. 07, 2020. [Online]. Available: <https://threatpost.com/threatlist-cryptominers-dominate-malware-growth-in-2018/139448/>

- [10] A. Zimba, Z. Wang, M. Mulenga, and N. H. Odongo, "Crypto mining attacks in information systems: An emerging threat to cyber security," *J. Comput. Inf. Syst.*, vol. 60, pp. 297–308, 2020.
- [11] A. Azmoodeh, A. Dehghantanha, M. Conti, and K.-K. R. Choo, "Detecting crypto-ransomware in IoT networks based on energy consumption footprint," *J. Ambient Intell. Humanized Comput.*, vol. 9, pp. 1141–1152, 2018.
- [12] R. Tahir et al., "Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2017, pp. 287–310.
- [13] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system callbased malware detection," in *Proc. Int. Symp. Softw. Testing Anal.*, 2012, pp. 122–132.
- [14] W. Ma, P. Duan, S. Liu, G. Gu, and J.-C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *J. Comput. Virology*, vol. 8, no. 1/2, pp. 1–13, 2012.
- [15] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, "Back-propagation neural network on Markov Chains from system call sequences: A new approach for detecting android malware with system call sequences," *IET Inf. Secur.*, vol. 11, no. 1, pp. 8–15, 2016.
- [16] D. Ceponis and N. Goranin, "Evaluation of deep learning methods efficiency for malicious and benign system calls classification on the AWSCTD," *Secur. Commun. Netw.*, vol. 2019, 2019, Art. no. 2317976.
- [17] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools Appl.*, vol. 78, no. 4, pp. 3979–3999, 2019.

- [18] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in linux containers," in Proc. IEEE Globecom Workshops, 2015, pp. 1–5.
- [19] H. Liang, Q. Hao, M. Li, and Y. Zhang, "Semantics-based anomaly detection of processes in linux containers," in Proc. Int. Conf. Identification Inf. Knowl. Internet Things, 2016, pp. 60–63.
- [20] A. Desnos, E. Petrova, A. Boulgakov, R. Neal, and Z. Mithra, "Flow-graph analysis of system calls for exploit detection," Tech. Discl. Commons, Jun. 2018. [21] M. Salehi and M. Amini, "Android malware detection using Markov Chain model of application behaviors in requesting system services," CoRR, vol. abs/1711.05731, 2017. [Online]. Available: <http://arxiv.org/abs/1711.05731>