# DESIGN AND DEVELOPMENT OF INTELLIGENT SECURITY MANAGEMENT SYSTEMS: THREAT DETECTION AND RESPONSE IN CYBER-BASED INFRASTRUCTURES

by

**Yahya Javed**

**A Dissertation**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Doctor of Philosophy**


School of Electrical and Computer Engineering

West Lafayette, Indiana

December 2021

# THE PURDUE UNIVERSITY GRADUATE SCHOOL
# STATEMENT OF COMMITTEE APPROVAL

**Dr. Arif Ghafoor, Chair**

School of Electrical and Computer Engineering

**Dr. Walid G. Aref**

Department of Computer Science

**Dr. Michael D. Zoltowski**

School of Electrical and Computer Engineering

**Dr. Ali A. Elghariani**

School of Electrical and Computer Engineering

**Approved by:**

Dr. Dimitrios Peroulis

To my parents for their love, support and prayers.

# ACKNOWLEDGMENTS

All praise is for Allah, the Almighty, who has bestowed upon me countless blessings and favors, and being the ultimate source of strength in my life.

I would like to express my utmost gratitude to my advisor Prof. Arif Ghafoor for his continued guidance and support throughout my PhD journey. His remarkable mentorship enabled me in developing my communication, technical and critical thinking skills. I consider myself extremely fortunate for having such an amazing advisor and I have always found something to learn from his distinguished research and academic career.

I would also like to thank Prof. Walid G. Aref, Prof. Michael D. Zoltowski and Dr. Ali A. Elghariani for serving in my doctoral advisory committee. Their invaluable comments and suggestions were of great help in the preparation of this dissertation.

I had the opportunity of working with many brilliant researchers: Anas Hazim Daghistani, Mosab Abdulaziz Khayat, Muhamad Felemban, Tawfeeq Shawly and Thamir Qadah. I would like to thank them and all of my colleagues of the Distributed Multimedia Systems Lab for their stimulating discussions and valuable feedback that helped me in my research.

I would like to thank my parents Muhammad Javed and Shadab Javed for their unconditional love and being the greatest source of inspiration in my life. I would also like to thank my siblings Bilal, Nayab and Hamza for always lifting my spirits. I would also like to extend my gratitude to my aunt Rubina Zaffar and uncle Mohammad Zaffar for providing me with a sense being at home during my stay in the United States.

In the end, I would like to thank the National Science Foundation and Northrop Grumman Mission Systems for supporting the research presented in this dissertation.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABSTRACT

Cyber-based infrastructures and systems serve as the operational backbone of many industries and resilience of such systems against cyber-attacks is of paramount importance. As the complexity and scale of the Cyber-based Systems (CBSs) has increased many folds over the years, the attack surface has also been widened, making CBSs more vulnerable to cyber-attacks. This dissertation addresses the challenges in post intrusion security management operations of threat detection and threat response in the networks connecting CBSs. In threat detection, the increase in scale of cyber networks and the rise in sophistication of cyber-attacks has introduced several challenges. The primary challenge is the requirement to detect complex multi-stage cyber-attacks in realtime by processing the immense amount of traffic produced by present-day networks. In threat response, the issue of delay in responding to cyber-attacks and the functional interdependencies among different systems of CBS has been observed to have catastrophic effects, as a cyber-attack that compromises one constituent system of a CBS can quickly disseminate to others. This can result in a cascade effect that can impair the operability of the entire CBS. To address the challenges in threat detection, this dissertation proposes PRISM, a hierarchical threat detection architecture that uses a novel attacker behavior model-based sampling technique to minimize the realtime traffic processing overhead. PRISM has a unique multi-layered architecture that monitors network traffic distributedly to provide efficiency in processing and modularity in design. PRISM employs a Hidden Markov Model-based prediction mechanism to identify multi-stage attacks and ascertain the attack progression for a proactive response. Furthermore, PRISM introduces a stream management procedure that rectifies the issue of alert reordering when collected from distributed alert reporting systems. To address the challenges in threat response, this dissertation presents TRAP, a novel threat response and recovery architecture that localizes the cyber-attack in a timely manner, and simultaneously recovers the affected system functionality. The dissertation presents comprehensive performance evaluation of PRISM and TRAP through extensive experimentation, and shows their effectiveness in identifying threats and responding to them while achieving all of their design objectives.

# 1. INTRODUCTION

The industrial digitalization and the recent focus on automation has made CBSs find their application in all major industries including internet-based services, retail, transport, defense and energy. Most of the CBSs perform mission critical operations that require high operational resilience. The rise in the number of cyber security incidents involving CBSs in the form of Advanced Persistent Threats (APTs) from state sponsored and other organized groups during the last couple of years has made it challenging to keep the CBS secure. The complexity and scale of the CBS has increased significantly over the past couple of years and as a consequence the attack surface has expanded proportionally. From a systems engineering perspective, the increase in system complexity makes it more challenging to provide service guarantees. From the security perspective, added complexity widens the attack surface [1]. Most of the cyber-attacks targeting the CBS are initiated through the network that includes malware propagation, infiltration, vulnerability scanning, botnets, Denial of Service (DoS) and Distributed Denial of Service (DDoS). Intrusion Detection Systems (IDSs) employed to detect malicious activities in the cyber networks have revealed their limitations in dealing with the present day threats as the median dwell time is reported to be of 30 days and internal detection rate is on a continuous decline [2]. Furthermore, the delay in detection and the corresponding delay in response enables the attacker to compromise the whole CBS once the initial penetration is successful. The functional interdependencies among different components of a CBS exacerbate the security risk as a cyber-attack that compromises one constituent system of a CBS can disseminate to others by normal system operations as well that can carry the malware or tampered data from one device to another. This can result in a cascade effect that can impair the operability of the entire CBS in a short period of time. These constraints and limitations in the threat detection and threat response of the current post intrusion security management systems entails a complete overhaul in the system design and operations.

## 1.1 Threat Detection in Cyber-based Systems

IDSs face three major challenges; an ever increasing attack complexity, astronomical increase in data traversing through the networks, and the need to detect attacks in realtime with in-depth information. To account for the growing attack complexity, modern IDSs evolved from simple string matching systems to sophisticated machine learning-based frameworks, making intrusion detection a computation intensive process [3]. To cater for the processing of massive data, there is no substantial change in the IDS computational process other than the introduction of more powerful processing hardware. However, according to Gilder's law the bandwidth grows at least three times faster than the compute power. While, compute power doubles every eighteen months (Moore's law), bandwidth doubles every six months [4]. This widening gap between the computational capacity and network bandwidth has made it immensely challenging for IDSs to process every packet in realtime and identify threats in a cost effective manner. Consequently, as the limited IDS resources are deployed at only few parts of the network, the chances of attackers penetrating the network without detection increase. At the same time, it is expected of the IDSs to identify threats with actionable information that can lead to a proactive response, contrary to binary classification like alerts that have little use for response operations[5]. These challenges necessitate an overhaul in the IDS architecture and the intrusion detection process.

Commercial and opensource IDSs for enterprise networks are generally deployed in a centralized orchestration. Some of these IDSs use multiple parallel processing nodes to handle large quantity of data [6]. The network traffic is captured by taping the link connecting the enterprise network to the internet which is then processed by the IDS detection module to find malicious packets [7], [8]. This design choice has several disadvantages including high cost of hardware that can process data in Gbps of data rates, limitations in scalability as the network grows over time, and inability to monitor internal network traffic that ignores attack sources other than the internet. In literature, there is a significant amount of work that proposes a distributed IDS design. While the proposed distributed IDS architectures address the problems associated with a centralized architecture, they introduce new issues in addition to ignoring some of the aforementioned challenges experienced by the IDSs. As discussed in

the related work section of the paper, some of the proposed distributed IDS architectures only focus on enhancing the traffic processing capabilities, but at the same time have the inability to detect complex multi-stage attacks, and do not possess intrusion prediction feature that can provide the response mechanism with enough information for a proactive response. On the other hand, the proposed IDSs with intrusion prediction capabilities present a conceptual distributed architecture that does not address the practical challenges associated with a distributed design. Among those challenges is the alert reordering issue that occurs when alerts are collected from distributed sources with different network latencies. This can cause prediction error in time series data-based prediction models that are generally employed to detect multi-stage attacks. Additionally, the proposed IDSs with intrusion prediction capabilities lack in proper implementation and evaluation in a distributed alert generation setup. This entails the need to develop an intrusion detection architecture that can process network traffic in parallel, predict the attack progression and detect complex multi-stage attacks, while addressing the challenges introduced by the distributed design.

Present-day IDSs process every packet traversing through their assigned region of observation in the network. For each packet or flow, a decision is made if they are malicious or benign. Most of the traffic in the network is benign, and even during an attack, the malicious traffic is a fraction of the total network traffic. Moreover, once an attack is detected, most of the malicious packets that follow are of the same continuing attack. These observations exhibit that there is room for optimizations in the way IDSs monitor traffic and detect threats. Network traffic sampling is a plausible approach to make intrusion detection much more efficient by directing only sampled traffic to the IDS. However, existing network sampling techniques are designed for network traffic engineering purposes and perform poorly for intrusion detection applications [9]. This necessitates the development of a network intrusion detection centric sampling scheme with the objectives of providing acceptable IDS accuracy, retention of IDS's ability to identify every attack and maximize attack detection speed by reducing the amount of traffic to be processed by the IDS.

## 1.2 Threat Response in Cyber-based Systems

CBSs integrate heterogeneous systems that have functional interdependencies. These integrated heterogeneous systems effectuate discrete functions and collaborate in an intricate fashion to perform the overall functionality of the CBS. CBSs are connected to the internet and employ cloud-based infrastructure for their computational requirements.Nowadays, CBSs are capable of performing multiple diverse functions as compared to the legacy systems. However, such capability can result in an increased system complexity. The functional interdependencies among the systems of CBS further exacerbates the security risk. An attack that is initiated at one sub-system of a CBS can disseminate to others. This can result in a cascade effect of compromising one system after other and eventually impairing the functionality of whole CBS in a short amount of time.

Some CBS security frameworks mitigate the security risk by promptly patching the system components for newly discovered vulnerabilities, in addition to implementation of strict authentication protocols for communication with the CBS [10]. This solution based on the target hardening concept is proving to be less effective as cyber-attack incidents reported over the years indicate a continuous increase in their sophistication. Another solution is based on the intrusion prevention approach, in which the security framework employs IDS that generates alerts when it identifies an anomaly in the working of CBS components. Intrusion prevention systems (IPSs), in addition to detection, can deploy some countermeasures, such as disconnecting the attacker's communication with the CBS, and halting the operation of affected CBS components [11].

The performance of IPSs is tightly coupled with the IDS efficiency. IDS is usually evaluated in terms of false positive rate (FPR), false negative rate (FNR) and detection delay time. High FPR indicates significant number of false alarms that can effectuate an unpredictable IPS behavior while high FNR implies an ineffective IPS. Detection delay time is often meted with less importance in performance evaluation of an IDS, but it has a significant effect on the operations performed by the IPS. Considering a scenario in which the intent of the attacker is to compromise as many devices of the CBS as possible. The attacker orchestrates the attack by systematically compromising one device after the other. The average time

taken by the attacker to compromise a device corresponds to the attack propagation speed, and the average time taken by the IDS to identify if a component has been compromised corresponds to the detection speed. If the attack propagation speed is more than the detection speed then the attacker can continue to compromise the CBS components while the IPS will try to catch-up until all of the target components are compromised. This "catch-up" process is evident when the strategy of the attacker is to outpace the system's response mechanism rather than to evade detection.

### 1.3 Primary Contributions

The contribution of the research presented in this dissertation has two parts: addressing the challenges in threat detection and addressing the challenges in threat-response for CBSs. To address the aforementioned challenges in threat detection, the dissertation introduces **PRISM: Performance-oriented Realtime Intrusion-detection and Security Monitoring** [12]. The fundamental design goals of PRISM are to detect complex multi-stage attacks in high bandwidth cyber networks with minimum processing overhead, provide intrusion prediction and holistic security assessment of the entire network in realtime, and should be scalable by comprising of a modular architectural construct. The distinct contributions of PRISM are as follows:

*1) Intrusion detection, prediction and security analysis:* PRISM incorporates a multi-layered structure that utilizes the services of its several internal systems to detect intrusions, predict attack progression and present a holistic security outlook of the system. The network traffic is monitored in a distributed fashion by dividing the network into different surveillance zones. Each surveillance zone is supervised by an IDS that reports only its local alerts. The alerts are correlated in a Hidden Markov Model (HMM)-based prediction system that first determines the type of multi-stage attack and then predicts its current stage. The prediction output and alerts from surveillance zones are then processed to compute and visualize several metrics for intrusion response facilitation.

*2) Threat-aware sampling:* PRISM uses a probabilistic sampling scheme to process more traffic from devices that have more likelihood of being attacked. The sampling scheme

utilizes a novel attacker behavior model-based ranking mechanism that ranks devices in the network according to their vulnerability and inter-device reachability information. Packets are sampled according to a sampling probability which is calculated using the rank-based score of the packet's source and destination device. Instead of processing every captured packet, only sampled packets are forwarded to the detection module of the IDS. Using the attacker behavior model-based sampling, PRISM strategically focuses on the traffic from those devices that are critical, and are more prone to be attacked, reducing processing overhead and the required computing resources.

*3) Alert stream management:* PRISM manages the alert streams from multiple surveillance zones by implementing an efficient alert stream management mechanism. The received alerts are sent to different systems of PRISM and are subjected to distinct treatments based on the requirement of their destination systems. For attack prediction, the alerts are reordered according to their IDS generated timestamps before being sent using a window-based stream processing implementation. For security analysis, the alerts are forwarded as soon as they are received without any processing to enable prompt security situation update.

*4) Experimentation and performance evaluation:* The performance of PRISM is evaluated through extensive experimentation on ISCX-2012 dataset [13]. The dataset consists of real traffic capture of several complex multi-stage attacks launched on a widespread target network for seven days. Experiments are designed to test the performance of all internal systems of PRISM in terms of multiple performance metrics. Results show that PRISM can maintain good prediction accuracy while reducing the processing overhead significantly in addition to being able predict attack progress notably early when studied under several experimental conditions. The security analysis capability of PRISM provides valuable utility by computing and visualizing cyber situational awareness-centric metrics.

To address the aforementioned challenges in threat-response, the dissertation presents **TRAP: Threat Response by Adaptive Partitioning** [14]. The major contributions of TRAP are as follows:

*1) Integrated security architecture:* TRAP combines the concepts of intrusion prevention and recovery to manage cyber-attacks while maintaining high operational availability of CBS.

*2) Adaptive partioning mechanism:* TRAP introduces a novel multi-level partitioning mechanism that partitions the CBS components based on their functionality and damage containment requirements. The partitions help to contain the damage by localizing the cyber-attack in the region of its inception. The response mechanism of the architecture effectively utilizes the partitions to counter attacks involving high attack propagation speeds as well.

*3) Experimentation and performance evaluation:* To demonstrate the performance of TRAP we have implemented Advanced Metering Infrastructure (AMI)-based cyber-attacks scenario by upgrading the source code of SecAMI simulator [15]. The performance of TRAP is evaluated in terms of operational availability, damage extent and Average Energy Load (AEL) consumption metrics.

## 1.4 Organization

The dissertation is organized as follows. In Chapter 2, related work corresponding to threat detection, threat prediction, network traffic sampling, attacker behavior modeling, threat response and system recovery is discussed. In Chapter 3, the design and internal workings of PRISM are explained with the discussion of how effectively PRISM addresses the challenges of threat detection in CBSs by a presenting comprehensive performance evaluation through extensive experimentation. In Chapter 4, the design of TRAP along with the functionality of its constituent systems is described with the demonstration of threat localization capabilities of TRAP in a practicable fast propagating cyber-attack scenario. Chapter 5 concludes the dissertation with the recommendations for future work.

# 2. RELATED WORK

This chapter overviews the work related to threat detection and threat response in CBSs. The related work has been classified into the following categories: 1) Security risk analysis, 2) Traffic sampling in network security, 3) Intrusion detection, 4) Intrusion prediction, and 5) Intrusion response and recovery.

## 2.1 Security Risk Analysis

Modeling of risk associated with the network using the vulnerability and topological information of the network has been extensively studied. The objective of most of these studies is to find out the relative likelihood of network nodes to be attacked, and to investigate which particular path is more likely to be taken by the attacker. Use of probabilistic attack graphs for security risk analysis has been proposed in [16]. Also, there are approaches that calculate the global or cumulative vulnerabilities associated with nodes using probabilistic attack graphs that can reveal the cumulative vulnerabilities of a network [17]. A Bayesian Decision Network (BDN)-based security risk management framework is proposed in [18]. The framework employs vulnerability exploitation probability, impact of vulnerabilities exploitation on network hosts in its risk assessment procedure using the temporal and environmental factors. In [19], network security risk assessment method is presented that analyzes the vulnerability information of different nodes of the network to compute the risk associated with each vulnerability. The vulnerability risk information is then used to generate access control lists to control the access to different nodes in the network for potential mitigation of threat. In [20], a risk assessment and optimization model for network security countermeasure problem is proposed. The model uses variables such as financial cost and impact risk to formulate countermeasures for network security incidents.

Modeling of attacker behavior using ranking approaches has been proposed as well. In [21], [22], [23] Google PageRank mechanism has been used to model the attacker behavior in terms of what attack states are more likely to be reached by an attacker in an attack graph. The goal is to reduce the complexity in analyzing attack graphs prior to undertaking appropriate countermeasures. In [24], a network risk management mechanism using attacker

profiling is proposed. The mechanism determines the risk associated to critical network assets by modeling how they can be compromised by an attacker. The behavior of the attacker is hypothesized to be based on the social behavior parameters such as skill level, tenacity and financial ability. The risk level is estimated using behavior based attack graphs that explore all possible attack paths to critical network assets. In [25], a model to analyze the cyber security risk is presented that combines the concepts of fault tree analysis, decision theory and fuzzy logic. The model evaluates the causes of failure of cyber security prevention methods and determines the vulnerability of the overall system. The focus of some contributions in the area of security risk analysis is to condense the attack graphs for the facilitation of security analysts. In [26], the concept of a core attack graph is proposed that illustrates the major paths an attacker can take in reaching the target in the network. Recently automation in vulnerability assessment through attack graphs has been focused, in [27], a tool and an end-to-end process is presented to identify and exploit vulnerabilities for cyber risk assessment.

### 2.2 Traffic Sampling in Network Security

The impact of sampling techniques designed for network traffic engineering on the performance of network intrusion detection is well studied. In [9], the suitability of four commonly used sampling techniques for intrusion detection applications has been investigated. The intrusion detection performance of random packet sampling, random flow sampling, sample-and-hold and smart sampling is compared. It has been shown that flow-based sampling performs better than packet-based sampling. Both sample-and-hold and smart sampling are found to be biased towards large volume flows, and are not suitable to detect several attack types. In [28], smart sampling and selective sampling techniques are evaluated for intrusion detection applications. Results demonstrate that selective sampling does not exhibit much bias towards large volume flows unlike smart sampling. In [29], a packet sampling algorithm for worm and botnet detection is presented. The proposed algorithm selects packets that contain first few bytes of a TCP connection by using a TCP connection tracking scheme and Bloom filters that store the connection states. In [30], an OpenFlow-based sampling mechanism is presented that uses a combination of stochastic and deterministic sampling

methods to tune the overall selection of packets for a specific application. For security related applications, it has been proposed that selecting long chain of consecutive packets is not required, instead packet selection with gaps is preferred. In [31], a sampling scheme designed for network intrusion detection has been proposed. The developed sampling mechanism is based on late and adaptive sampling process. The late sampling process extracts flow features and calculate flow statistics. Adaptive sampling process then selects outlier flows with more preference as compared to flows with similar flow statistics. Recently, machine learning-based sampling techniques have been employed for network intrusion detection applications. In [32], a hierarchical clustering-based sampling technique is proposed that uses structural and temporal features of suspicious flows in a two-step hierarchical clustering algorithm. In [33], the impact of sampling on anomaly detection is studied by comparing the performance of four sampling techniques. Weighted Round Sampling (WRS), deterministic sampling, Simple Random Sampling (SRS) and chain-sampling algorithms are evaluated. It has been shown that WRS performs better than the other evaluated sampling techniques. The dissertation proposes a novel threat-aware sampling scheme that distinguishes from the existing sampling techniques by employing an attacker behavior model-based ranking mechanism that uses the vulnerability and inter-device reachability information to sample more traffic from the devices that are more likely to be compromised by the attacker.

## 2.3 Intrusion Detection

The first known intrusion detection model was presented by D.E. Denning [34] afterwards a number of contributions has been made in the field. IDSs can be classified into two major groups based on the methodology employed to detect intrusions [35]. The first group is known as signature-based IDSs in which the IDS maintains signatures or patterns of known cyber-attacks that have occurred previously. The rules of these systems are usually created by the security analysts and each time a new attack type is discovered, a new signature is created and an update is carried out for the IDS. Signature-based IDSs continuously observe phenomenon of interest in the system and match their observations with the signature library. When a match occurs, an alert is raised by the signature-based IDS giving specific information about

what violations occurred that made the IDS to raise alarm. Signature-based IDSs are most widely used and perform well in identifying the details of the cyber-attack that can be used for intrusion response. The drawbacks of signature-based IDS is that they are unable to detect zero day cyber-attacks. The second group is known as behavior or anomaly-based IDSs in which the IDS models the normal behavior of the system and observes the behavior of the system for specific modeled features. When the IDS observes deviation from the normal then it raises an alert. The alerts in an anomaly-based IDS are usually binary, meaning that they only inform about the presence of malicious activities, nothing more, therefore the use of anomaly-based IDSs in automated intrusion response is limited. Additionally, if the normal behavior model is less representative of the true normal behavior of the system then any previously unseen normal activity might be flagged as anomalous. The benefits of using anomaly-based IDSs is that they can detect zero-day attacks [36].

The architecture of IDSs is another important aspect by which they are classified. Many IDS architectures have been proposed over the years with a certain IDS architecture being dominant for a particular network type. For enterprise networks a centralized IDS deployment is the most common in which the IDS scrutinises all traffic passing through the external routers that connect the enterprise network to internet [37]. Internet of Things (IoT) networks are generally protected by distributed IDS as there is no central location from where data of the whole network can be monitored [38]. A distributed IDS architecture for enterprise networks is proposed in [39] that has IDS components distributed into different regions of the network. Each regional detector reports malicious activities of its sphere to a global detector that uses Sequential Hypothesis Testing (SHT) to determine whether the whole enterprise network is under attack. A distributed IDS for smart grid networks is presented in [40]. In the proposed system there is an intelligent Analyzing Module (AM) at every layer of the smart grid. Multiple AMs are embedded in Home Area Networks (HANs), Neighborhood Area Networks (NANs) and Wide Area Networks (WANs). AMs use Support Vector Machine (SVM) and Artificial Immune System (AIS) to detect and classify malicious traffic. AMs are trained using the data relevant to their level and have the capability to communicate with other AMs at their level and other levels to improve detection. In [41], the authors propose a fully distributed architecture in which the IDS has several IDS nodes.

Each sub-network is observed by a sentry node that communicates with the sentry nodes of other sub-networks as well. In addition to sentry nodes there are a higher level configuration entities called the survey nodes that work as a long term memory to cache rules, sub-network states and representations of connections. A collaborative anomaly detection framework has been proposed in [42] that uses network virtualization techniques to implement a global situation detection and local behavior detection model. The framework uses Hidden Markov Random Field (HMRF) to model network behavior and spatial Markovianity to model the network behavior's spatial context. Recently, the use of blockchain technology in distributed IDS design has become popular. In [43], a distributed IDS has been proposed that uses blockchain and smart contracts to mitigate threats on collaborating IDSs on different cloud networks. None of the abovementioned distributed IDSs, unlike PRISM, have the capability to identify complex multi-stage attacks. Additionally, the discussed distributed IDSs lack the ability to predict the attack progression. Moreover, they do not address the practical issues associated with the distributed design such as the effect of alert reporting delays due to network latencies when the alerts are gathered from various sources distributed all over the network for holistic threat assessment.

### 2.4 Intrusion Prediction

Intrusion prediction, attack projection, attack intention recognition and network security situation forecasting has been a focus of the research community for quite some time and several techniques have been proposed [44]. Many different approaches and techniques have been proposed over the past couple of years for prediction of multi-stage attacks. These techniques include the use of Markov Models [45], Bayesian networks [46], clustering [47], neural networks [48] and attack graphs [49]. The dissertation focuses its discussion on intrusion prediction using machine learning models only, specifically HMM-based intrusion prediction.

In [50], an SVM-based network security situation assessment and prediction method is introduced. The presented mechanism analyzes the IDS logs of the network that contain the security incident information from heterogeneous sources at the three levels of service, host

and network. The mechanism then employs SVM to predict the security situation of the network. Recently, the use of neural networks for the prediction of network security situation has become popular. In [51], a Long Short-Term Memory (LSTM)-based network security situation prediction method has been proposed. The method uses the LSTM to predict the network security data and then uses XGBoost to determine the network security situation using the predicted data. In [52], another LSTM-based technique to predict cyber-attacks is presented. The proposed technique uses an architecture with an LSTM layer together with a Multi-Layer Percepton (MLP) to predict the type of attack likely to occur. In [53], a network security situation assessment method is proposed that is based on adversarial deep learning. The proposed method is constructed by combining Deep Autoencoder (DAE) and Deep Neural Network (DNN). The Deep Autoencoder-Deep Neural Network (AEDNN) model uses the DAE network for feature learning and the DNN network to identify attacks. After the attack identification, network security situation is determined by conducting the impact assessment for each type of attack.

The use of HMMs for network intrusion detection is not new, in [54]–[58] anomaly detection systems using HMMs are presented. Recently, HMMs have been extensively used for network intrusion prediction. An intrusion prediction and prevention system has been presented in [59] that uses HMM to predict the next step of an attacker. The intrusion prediction information is then used for intrusion prevention operations according to a set of predetermined rules. In [60] data mining driven algorithm is proposed that mines the stream of IDS alerts for attack scenario extraction. An HMM-based alert correlation system is then introduced that predicts the next class of attacks to be launched by the intruder. Frameworks to predict multi-step network attacks using HMMs are proposed in [61], [62]. In [63], HMM-based architectures have been proposed that predict the attack stage when several multi-step attacks are in action simultaneously. Lately, to address the issue of limited availability of network intrusion detection datasets, a transfer learning-based approach was proposed that trains the HMM model using a labelled dataset and then adapts the model to a target unlabelled dataset [64]. The aforementioned intrusion prediction techniques only offer a centralized approach to process network traffic, while the dissertation, on the other hand presents an intrusion prediction architecture with distributed traffic processing. More-

over, the dissertation addresses the challenges associated with the distributed design by introducing an efficient alert stream management mechanism.

## 2.5 Intrusion Response and Recovery

The design and development of Intrusion Response Systems (IRSs) has received less importance than the IDS from the academic research community and the industry. The frequency and magnitude of cyber-attacks seen over the last couple of years compelled the security community to undertake the task of incident response automation as manual handling of intrusions is becoming increasingly difficult. As a result, practical IRSs have started to appear with automated response capabilities. IRSs can be generally categorized into three groups: notification systems, manual response systems and automated response systems [65]. Notification systems just collect alerts from the IDS and present it to a human administrator to respond to the notified threat. Manual response systems have some preconfigured response actions to the type of alerts received by the IDS for the human administrator to execute. Automated response systems receive the alerts from IDS, assess the threat, prepare and execute the response in a fully automated fashion without any human intervention. In [66], the design of an intrusion tolerant system is presented with the idea of distributing the system into different security domains in such a way that it is hard by the attacker to pass these security domains by the help of firewall and LAN configurations. In [67], an automated response framework is presented by using an Intrusion Detection and Isolation Protocol (IDIP). According to IDIP, the response nodes are distributed into IDIP communities which are further divided into IDIP neighborhoods. When an attack is detected by an IDIP node its information is distributed to other neighboring nodes for generating response. Though the concept has many merits but the catch-up problem cannot be catered by IDIP alone. Moreover, the criteria for the creation of communities and neighbourhoods is also not well defined. In [68], a tracing approach is presented to identify the chain of compromised machines that was used to compromise the target host and once the chain is identified then the response can be applied. Their approach doesn't solve the significant issues in intrusion response like damage containment and automated intrusion response. In [11], a

game-theoretic formulation of ideal response strategies once an intrusion is detected in the network is presented. In their work no proper concept of damage containment is introduced nor an automated response mechanism is proposed. Recently, the utilization of Markov Decision Process (MDP)-based intrusion response mechanisms for network security is becoming popular. In [69], an IRS that uses a probabilistic model on the MDP framework is proposed. The model is able to plan optimal response actions with the objective of long-term system security. In [70], a Partially Observable Markov Decision Process (POMDP)-based intrusion response approach for cyber networks is proposed. The proposed model hampers the progression of an attacker while minimizing the negative impact to system availability. The progression of attacker is measured by embedding a state space on the condition dependency graph which is an attack graph that models dependencies between attacker's capabilities and exploits. The countermeasures of the defender is in the form of blocking some of the exploits from being carried out by the attacker. The attacker's strategy in traversing through the network is driven by its private strategy which is dependant on the defender's actions. The basic problem with all MDP-based models is the generation of very large state space which makes it exhaustive to compute the optimal response. Also, the aforementioned MDP-based intrusion response models do not address the threat containment problem for fast progressing cyber-attacks. The dissertation presents an intrusion response and recovery architecture that uses a novel adaptive partitioning mechanism to localize fast progressing attacks while recovering the affected system components simultaneously.

# 3. PRISM: A HIERARCHICAL INTRUSION DETECTION ARCHITECTURE FOR LARGE-SCALE CYBER NETWORKS

This chapter presents PRISM, an intrusion detection architecture that monitors network traffic distributedly in different surveillance zones, samples the traffic before sending it to the IDS, combines the alert streams from different surveillance zones while correcting any discrepancies in the order of the received alerts, processes the combined alert sequence in an HMM-based prediction engine to predict the progression of the attack, and provides essential security analysis of the system. We start by discussing the background in Section 3.1. The system model and architecture is presented in section 3.2. Experimental setup and implementation details are discussed in section 3.3. Finally, performance evaluation of PRISM is presented in section 3.4.

## 3.1 Background

PRISM is built by utilizing the concepts of attacker behavior modeling, HMM and stream processing. In this section, the basics of the underlying methods and models used in PRISM are presented.

### 3.1.1 Attacker Behavior Modeling

Attacker behavior modeling is a widely used technique to evaluate the security of enterprise networks [71]. PRISM models the attacker behavior by extending the Google PageRank mechanism. PageRank is flexible enough to model a diverse set of cases and its employment in network security related graph problems is not uncommon [22], [23]. The behavior of a random web-surfer is modeled by PageRank who starts at a web-page and keeps on clicking links until gets bored and starts at another web-page. A web-page has higher rank if many pages link to it, and a page is important if it has a high rank and has few links to other pages. To model the probability of the web-surfer breaking the chain of clicking links and starting from a new random web-page, a damping factor is introduced which essentially asserts that

long chains of page clicking is unlikely [72]. PageRank iteratively calculates the rank of each page using Eq. 3.1.

$$PR(U) = 1 - \delta + \delta \sum_{\mathrm{j} \in In(U)} \frac{PR(W_{\mathrm{j}})}{Out|W_{\mathrm{j}}|} \tag{3.1}$$

### 3.1.2 Hidden Markov Model

Markov chains are used to determine the probability of a sequence of events that are observable. However, there are many instances in which the events of interest are not directly observable or hidden. A discrete first-order HMM is a doubly stochastic model that accommodates both observed and hidden events. An HMM is characterized by the following elements [73].

1) The set of N states represented by $S = \{s_1, s_2, .., s_N\}$, where state at time $t$ is $q_t$.

2) A set of M distinct observation symbols per state denoted as $V = \{v_1, v_2, .., v_M\}$.

3) Transition probability matrix $A_{NxN}$ where $a_{\mathrm{ij}}$ is the probability of shifting from state $s_{\mathrm{i}}$ to state $s_{\mathrm{j}}$.

$$a_{\mathrm{ij}} = P(q_{t+1} = s_{\mathrm{j}} | q_t = s_{\mathrm{i}}), \qquad \mathrm{i, j} \in [1, N] \tag{3.2}$$

4) Emission probability matrix $B_{NxM}$, where $b_{\mathrm{i}}(k)$ is the probability of an observation $v_k$ being generated in a state $s_{\mathrm{i}}$.

$$\begin{aligned} b_{\mathrm{i}}(k) = P(v_k \ at \ t | q_t = s_{\mathrm{i}}), \qquad & \mathrm{i} \in [1, N] \\ & k \in [1, M] \end{aligned} \tag{3.3}$$

5) Initial probability vector $\boldsymbol{\pi} = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, .., \boldsymbol{\pi}_N\}$, where $\boldsymbol{\pi}_{\mathrm{i}}$ is the probability of Markov chain initializing at state $s_{\mathrm{i}}$.

$$\boldsymbol{\pi}_{\mathrm{i}} = P(q_1 = s_{\mathrm{i}}), \qquad \mathrm{i} \in [1, N] \tag{3.4}$$

HMM-driven prediction systems use the model $\lambda = (S, V, A, B, \boldsymbol{\pi})$ and observation sequence $O = \{o_1, o_2, .., o_T\}$ to effectuate various tasks. Note that observations are IDS alerts

in our case and we use the terms observation sequence and alert sequence interchangeably. The use of HMMs in practical applications is characterized by three fundamental problems. First, how to determine $P(O|\lambda)$, the probability of observation sequence given the model. Second, how to discover the best state sequence $Q = q_1, q_2, .., q_T$ given the observation sequence and model. Third, how to learn the model parameters $A$ and $B$ that maximize $P(O|\lambda)$ given the observation sequence and the set of states. The first problem is solved using the forward algorithm, the second problem is solved using the Viterbi algorithm and for the third problem the standard solution is unsupervised learning-based Baum-Welch algorithm, but in our implementation we follow a supervised learning approach discussed in Section 3.3 of the dissertation.

### 3.1.3 Stream Processing

In the event of an attack, a continuous stream of alerts is generated by the IDSs. PRISM employs stream processing techniques to perform certain data manipulation operations on this stream of alerts. In the stream processing context, the terms bounded and unbounded refers to the finite and infinite data, respectively. Performing data processing operations like sort, aggregation etc. on unbounded data causes semantic problems, therefore, it is important to have some notion of bounds on the unbounded data. Windowing is a process of dividing data into finite blocks that can be processed as a group. The process of windowing is essential to analyze unbounded data and it has been proven to be useful for bounded data as well in certain scenarios. Windowing is generally time-based or count-based, however, there are several other implementations as well. As the names suggest, a time-based window defines a data block on the data stream with respect to a time period, while grouping a certain number of data items together forms a count-based window. The three main window types are tumbling, sliding and sessions. Tumbling windows are non-overlapping windows defined by constant window size and a data item cannot be the member of more than one window. Sliding windows are defined by a fixed window size and a slide period which may be less than the window size implying that the windows can overlap. In a sliding-windows-based implementation, a data item can be part of multiple windows. Sessions are windows

that are defined over a subset of data using a timeout gap. The events that occur within a timeout are combined to form a session [74]. PRISM uses a custom implementation of the windowing process employing both time-based and count-based windows to manage the flow of alerts streams.

## 3.2 System Model and Architecture

.PRISM is depicted in Fig. 3.1 along with its five constituent systems: Threat-Aware Sampler (TAS), Zonal IDS, Alert Stream Manager (ASM), Prediction Engine, and Security Analyzer. TAS and Zonal IDS implement the distributed security monitoring functionality of PRISM and are replicated among multiple surveillance zones. In Fig. 3.1 only one surveillance zone is shown for lucidity. The demarcation of surveillance zones can be done based on the network layout, traffic volume distribution and amount of available zonal IDS resources. The objective is to distribute the traffic equally among surveillance zones with the consideration of network administrative constraints. The number of available zonal IDS resources determine the number of surveillance zones. ASM, Prediction Engine and Security Analyzer implement the integrated security analysis functionality of PRISM. The detailed information about the design and working of each component of PRISM is discussed as follows.

### 3.2.1 Threat-Aware Sampler

TAS is deployed in every surveillance zone along with the zonal IDS. Mirrored traffic from the router of the observation zone is forwarded to the TAS which samples traffic in realtime and directs it to the zonal IDS. TAS has offline and online modes of operation. In offline mode, TAS performs three operations: vulnerability assessment, network reachability graph generation and vulnerability ranks and sampling probabilities computation. In online mode, TAS carries out the operation of realtime probabilistic sampling. The architecture of TAS is shown in Fig. 3.2 with its interaction to a general network and IDS. The functioning of the internal modules of TAS are discussed in detail.

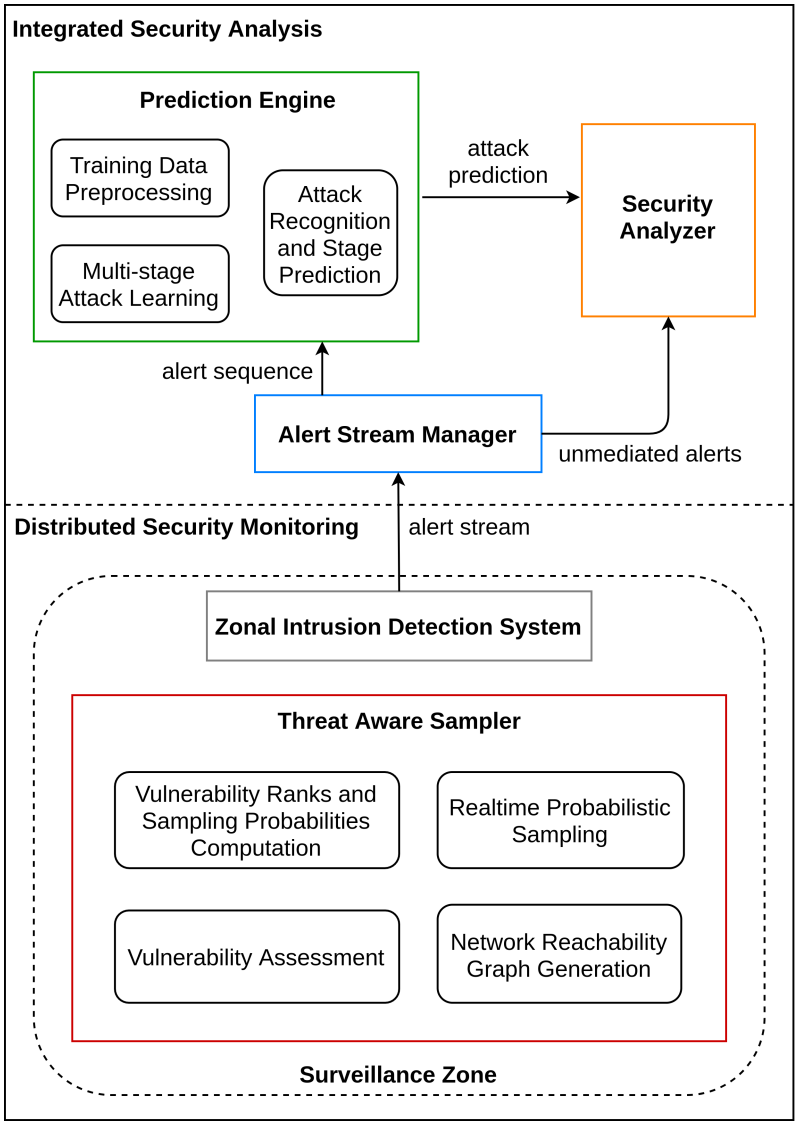


**Figure 3.1.** PRISM and its constituent systems.

### Vulnerability Assessment

The vulnerability assessment module of TAS performs vulnerability scan of each device in the network to determine their Common Vulnerability Scoring System (CVSS) scores [75]. CVSS is a popular measure of assessing the severity of a device's vulnerabilities by assigning a score out of 10. CVSS scores are computed using the information of three metric groups: base, temporal and environmental. For base metric group, information of five exploitability metrics and three impact metrics is required to calculate the score of

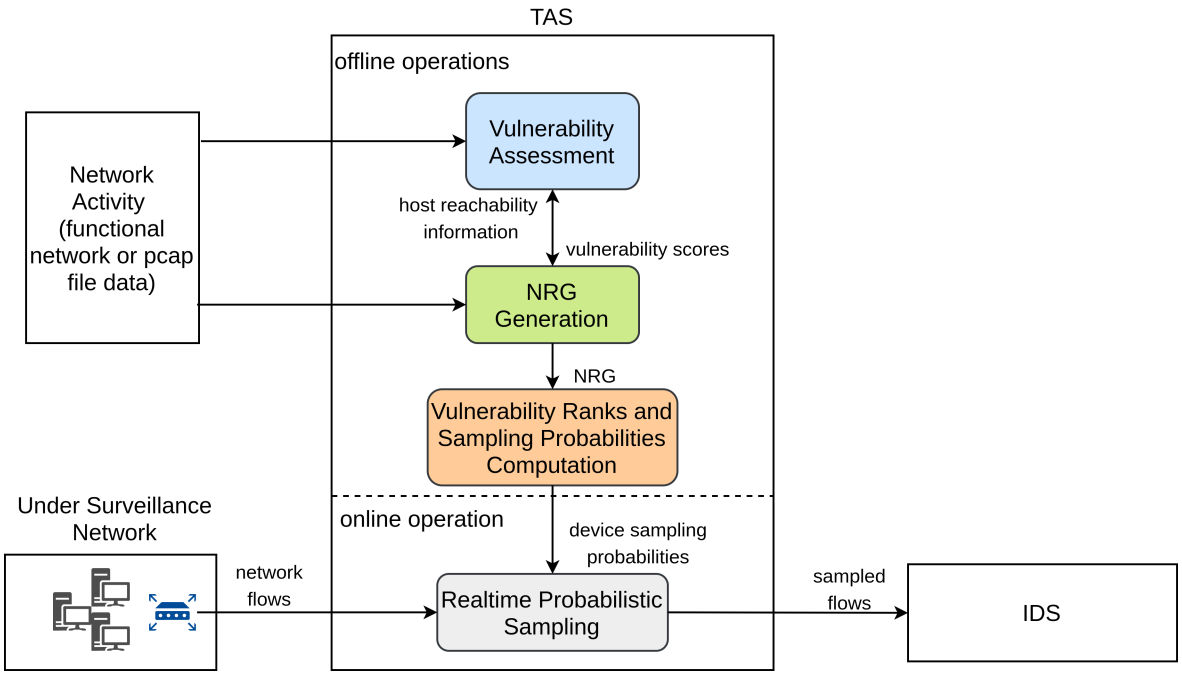


**Figure 3.2.** TAS architecture and its interaction with the environment.

a vulnerability. The vulnerability score for a device can be obtained by considering the CVSS scores of all vulnerabilities identified in a device. There are many commercial and opensource tools available that can scan the network and assess the vulnerabilities of each device [76], [77]. TAS has the flexibility to operate with any proprietary or opensource CVSS compatible network vulnerability scanner. The vulnerability assessment procedure adapted for experimentation is discussed in Section 3.3 of the dissertation.

### Network Reachability Graph Generation

TAS employs the Network Reachability Graph (NRG) to develop the attacker behavior model that is used to rank network devices according to their likelihood of being compromised. NRG$< D, E >$ is a directed mutligraph with its vertices corresponding to the devices in the network represented by $D = \{d_1, d_2, ..., d_n\}$. The parallel edges of NRG correspond to the end-to-end communication between devices on different ports and are represented by $E$, a multiset of edges (ordered pairs of vertices). A device in the network is identified by its i$p$ address, and for each device, the vulnerability score $vs$ computed by the vulnerability

assessment module is also maintained. The communication among network devices is in part determined by firewall policies, network configuration and authentication/permissions within devices. A sample NRG is illustrated in Fig. 3.3. NRG can be generated using any network mapping tool like Nmap [78]. Another method of generating NRG is by mapping the flow information of monitored network activity over time. We have employed the latter method that is discussed in Section 3.3 of the dissertation.

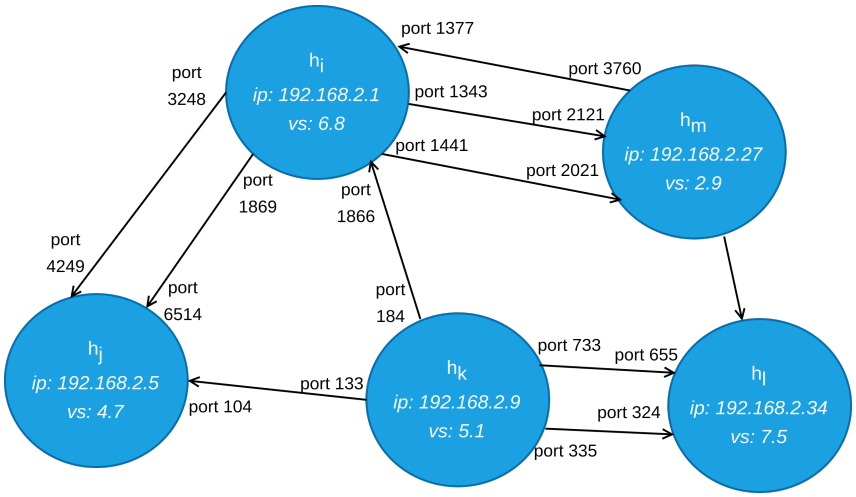


**Figure 3.3.** A sample 5-device NRG.

### Vulnerability Ranks and Sampling Probabilities Computation

The core idea behind threat-aware sampling is to intelligently sample traffic by modeling the attacker behavior that can rank network devices according to their likelihood of being targeted by the attacker. The proposed attacker behavior model captures the potential actions of an attacker who has a specific target and tries to penetrate the network through any accessible device. From the compromised devices, the attacker tries to strike every other device until the attainment of target. Fig. 3.4 shows an example attack in which the attacker compromises devices $d_2$, $d_6$, and $d_8$ to attack $targetB$ by exploiting vulnerabilities $\vartheta^a_{d_2}$, $\vartheta^x_{d_6}$, $\vartheta^y_{d_8}$ and $\vartheta^z_{targetB}$, respectively. The attacker behavior can be modeled by extending the random surfer model with security-centric semantics. A device has a high vulnerability rank if it has high vulnerability score and has wide attack surface i.e. several devices can

communicate with it on multiple ports. A device's vulnerability rank is also high if the devices communicating to it have high vulnerability scores. The damping factor in the attacker behavior model corresponds to the fact that it is unlikely for an attacker to take a longer path to reach target if a shorter path exists. The extended PageRank formalism that captures the vulnerability transference effect of inter connected devices is presented in Eq. (3.5). Where, $\tau_{\mathrm{i}}$ represents the vulnerability transference effect of a device $d_{\mathrm{i}}$, $\delta$ is the damping factor, $In(d_{\mathrm{i}})$ is the in-set of device $d_{\mathrm{i}}$ i.e. it is the set of devices transmitting to $d_{\mathrm{i}}$, $\rho_{\mathrm{j}}$ is the vulnerability rank of device $d_{\mathrm{j}}$ which is one of the devices in the in-set of $d_{\mathrm{i}}$, $|Out(d_{\mathrm{j}})|$ is cardinality of the set of outgoing edges from device $d_{\mathrm{j}}$, and $|Out(d_{\mathrm{j}}, d_{\mathrm{i}})|$ is cardinality of the set of edges from device $d_{\mathrm{j}}$ to $d_{\mathrm{i}}$.

$$\tau_{\mathrm{i}} = 1 - \delta + \delta \sum_{\mathrm{j} \in In(d_{\mathrm{i}})} \rho_{\mathrm{j}} \frac{|Out(d_{\mathrm{j}}, d_{\mathrm{i}})|}{|Out(d_{\mathrm{j}})|} \tag{3.5}$$

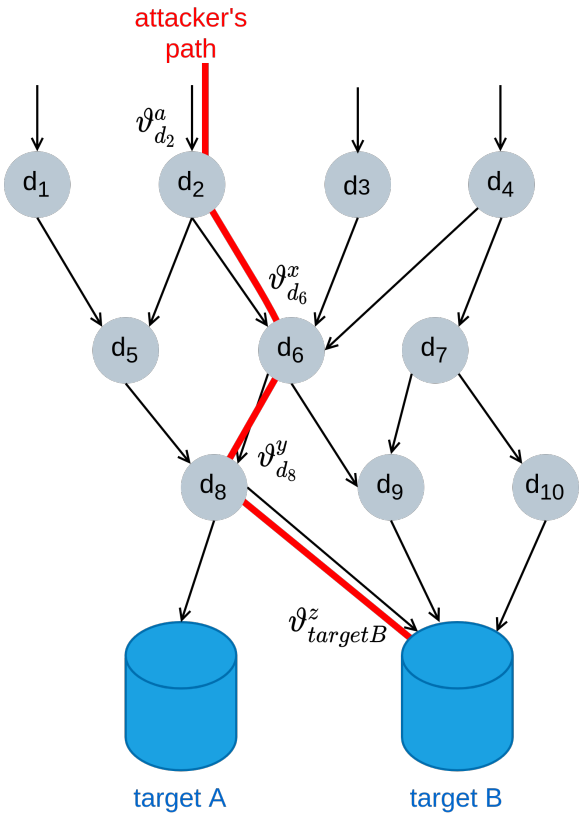


**Figure 3.4.** Illustration of attacker's movement in the network.

A device receives its vulnerability transference effect from the devices that can communicate with it and the devices that can communicate using multiple connections (port pairs) contribute more to the vulnerability transference. In vulnerability rank computation of a device, the device's self vulnerability contributes to its rank in addition to the contributions from the neighboring transmitting devices. This conforms to the semantics of the proposed attacker behavior model that a device is likely to be compromised by an attacker if it has high vulnerability and is surrounded by devices that have high likelihood of being targeted. It can be argued that why not simply use the vulnerability of a device as a measure to estimate its likelihood of being compromised. The answer is in the fact that the position of a device in the NRG affects its likelihood of being reached by the attacker. If a highly vulnerable device is surrounded by devices with low vulnerabilities, the chances of the attacker reaching that device become less. Similarly, if a device with low vulnerability is surrounded by devices with high vulnerabilities then the likelihood of that device getting compromised increases. The rank and sampling probability computation module of TAS uses Algorithm 1 to perform its operations. The algorithm produces two vectors $R = \{\rho_1, \rho_2, ..., \rho_n\}$ and $\Psi = \{\psi_1, \psi_2, ..., \psi_n\}$ corresponding to the vulnerability ranks and sampling probabilities of $n$ devices in the network. The values of $epochs$ and $thresh$ regulate the convergence conditions of the algorithm. For each device, first the vulnerability transference effect is computed and then it is combined with the vulnerability score to obtain the vulnerability rank value of the device. Finally, the vulnerability rank values are used to calculate the sampling probabilities by employing the softmax function. Algorithm 1 is not computationally intense and it has been shown that PageRank for millions of web-pages can be computed in few hours on a medium size workstation [72].

### Realtime Probabilistic Sampling

Once the vulnerability ranks and sampling probability computation module produces the vulnerability ranks $R$ and corresponding sampling probabilities $\Psi$ of all devices in the network, flows (or packets) are sampled in realtime before being sent to the zonal IDS. Let

**Algorithm 1** Vulnerability Ranks and Sampling Probabilities Computation

**Input:** $NRG < D, E >, \delta, epochs, thresh$
**Output:** $R = \{\rho_1, \rho_2, ..., \rho_n\}, \Psi = \{\psi_1, \psi_2, ..., \psi_n\}$

$R \longleftarrow$ *vulnerability scores of* $n$ *devices in* $D$
$\Psi \longleftarrow$ *zero vector of length* $n$
**for** i $\in [1, epochs]$ **do**
  $R_{last} = R$
  **for** $d_\text{i} \in D$ **do**
    /*vulnerability transference rank from Eq. 3.5*/
    $\tau_\text{i} = 1 - \delta + \delta \sum_{\text{j} \in In(d_\text{i})} \rho_\text{j} \frac{|Out(d_\text{j}, d_\text{i})|}{|Out(d_\text{j})|}$
    $\rho_\text{i} = \rho_\text{i} + \tau_\text{i}$
  **end for**
  /*calculate change in rank values using L1 norm*/
  $err = sum(abs(R)) - sum(abs(R_{last}))$
  **if** $err \leq thresh$ **then**
    break
  **end if**
**end for**
**for** j $\in [1, n]$ **do**
  /*softmax function to compute sampling probs.*/
  $\psi_\text{j} = exp(\rho_\text{j})/sum(exp(R))$
**end for**

$X_\text{ij}$ be a flow from device $d_\text{i}$ to $d_\text{j}$, then the probability of flow $X_\text{ij}$ of being sampled is provided in Eq. 3.6.

$$P(X_\text{ij}) = \alpha\ \psi_\text{i} + \beta\ \psi_\text{j} + \gamma\ \frac{v_\text{ij}}{\omega} \tag{3.6}$$

Where, $\psi_\text{i}$ and $\psi_\text{j}$ are the sampling probabilities of devices $d_\text{i}$ and $d_\text{j}$, respectively. The volume of the flow $X_\text{ij}$ is represented by $v_\text{ij}$ and $\omega$ is the flow volume threshold corresponding to maximum practicable volume of a flow in the network. The parameters $\alpha$, $\beta$ regulate the relative influence of the of source and destination device vulnerability ranks on the overall probability of sampling the flow, while $\gamma$ controls the contribution of flow volume on the overall sampling probability of the flow. It is to be noted that $\alpha, \beta, \gamma \in [0, 1]$ and $\alpha + \beta + \gamma = 1$. The parameters $\alpha$, $\beta$ and $\gamma$ are configurable, and their different values implement two distinct versions of TAS. The primary version of TAS is implemented in our experiments with $\alpha = 0.5$, $\beta = 0.5$ and $\gamma = 0$, that makes the sampling probability of the

flows invariant to flow volume. The other version of TAS we refer to as dynamic TAS (d-TAS) is implemented in our experiments by setting $\alpha = 0.25$, $\beta = 0.25$ and $\gamma = 0.5$. The inclusion of volume in determining the sampling probability of a flow serves two purposes. First, it is helpful in identifying certain attack types in which the malicious traffic is generally carried by flows of large volumes. Second, as discussed earlier, it serves a major purpose of being a dynamic parameter that improves the robustness of TAS against evasive strategies employed by an adversary that can make the primary version of TAS less effective. Based on the information possessed by the attackers, we distinguish them in two categories: conventional attacker and adept attacker. A conventional attacker doesn't have significant information about the target network and acquires information while traversing through the network. The conventional attacker starts the attack from a random host and then from there, if successful, tries to compromise every possible host until the attainment of target. Adept attacker, on the other hand, is assumed to have some prior knowledge about the network host vulnerabilities, and using that information, the adept attacker tries to compromise less vulnerable hosts. This strategy is adopted by the adept attacker to evade detection as intuitively high vulnerability parts of the network are monitored more strictly. The proposed attacker behavior model through the extension of PageRank mechanism with security-centric semantics captures the potential actions of the conventional attacker competently. However, the adept attacker can make the proposed model less effective by not compromising the most vulnerable hosts to reach the target, that is, the next host to be targeted by the adept attacker from a particular compromised host is not necessarily the most vulnerable one even if this action requires more effort. To counter these evasive strategies employed by the adept attacker, we have included the flow volume in computing the sampling probability of a flow. Flow volume is a dynamic parameter that is distinct for each flow that appears in the network, making the proposed TAS more resilient. Note that we assume the adept attacker is unaware of how TAS works but has general intuition that hosts with higher vulnerability tend to be under high surveillance. It is possible to employ parameter estimation techniques on historical intrusion detection data to find optimal values of $\alpha$, $\beta$ and $\gamma$. The computation complexity of realtime probabilistic sampling is minimal as vulnerability rank scores of hosts

are determined in the offline mode and extraction of the flow volume information is not computationally demanding.

### 3.2.2 Zonal Intrusion Detection System

Zonal IDS receives sampled traffic from TAS, finds malicious activities in its surveillance zone and sends alerts to the ASM. IDSs can be either signature-based or anomaly-based, depending on the methodology they employ to detect intrusions. Signature-based IDSs maintain signatures or rules that are used to identify known cyber-attacks. These rules are generally created by the security analysts, but they can also be learned using supervised machine learning techniques. Signature-based IDS are widely used and perform well in identifying malicious traffic with considerable detail, but are unable to detect zero-day attacks. On the other hand, anomaly-based IDSs model the normal behavior of the network, and raise alerts when a traffic pattern is observed that deviates from the normal behavior. Anomaly-based IDSs can detect zero-day attacks, but they provide little information about the detected malicious activity. Anomaly-based IDSs are mostly developed using unsupervised machine learning techniques, and their use as a stand-alone IDS solution is limited. PRISM can incorporate any IDS provided that the alerts contain information regarding the vulnerability exploited, devices impacted and intrusion type. A signature-based IDS is more compatible with other systems of PRISM, however, a hybrid signature/anomaly-based IDS can be used as well. In its current implementation, PRISM uses the opensource signature-based Snort IDS [7].

### 3.2.3 Alert Stream Manager

ASM receives streams of alerts from the Zonal IDSs located in different surveillance zones, and forwards it to the Security Analyzer and Prediction Engine. The alerts are sent to the Security Analyzer as soon as they received. For security analysis functions, the order in which the alerts are received is not critical, but to update security information of the system promptly, it is essential to forward the alerts as soon as they are received. Between reception and forwarding of alerts to the Prediction Engine, ASM performs a set of important

operations that ensures the alerts being forwarded to the Prediction Engine are in the same order as they are generated at their respective Zonal IDSs. The alerts can be reordered due to network latencies, unsynchronized clocks of Zonal IDSs and data transmission over a non-order-preserving channel. ASM addresses the alert reordering issue due to alerts being reported with different network latencies from different Zonal IDSs. To rectify this problem, ASM processes the unbounded and unordered streams of alerts by using a window-based stream management procedure. The essence of this procedure is the fact that we can estimate certain bounds on the network latencies. Bounds on the network latencies between Zonal IDSs and ASM can be computed from network characteristics or historical data that keeps track of the alert generation and reception time difference [79].

Consider $Z_1, Z_2, .., Z_n$ Zonal IDSs with alert reporting latencies upper-bounded by $L_1, L_2, .., L_n$, respectively. The actual maximum alert reporting latency is denoted by $\Delta_{max} = max(L_1, L_2, .., L_n)$, where the estimate of maximum alert reporting latency is represented by $\mu_{max}$. Each Zonal IDS generates a stream of alerts where an alert has a number of attributes, among which the alert generation timestamp $ts$ is crucial for ASM operations. The alerts are received by the alert buffer $Y = \{o_1, o_2, .., o_t, ..\}$, where $o_1$ is first alert in the buffer and $o_t$ is the alert received at a certain time $t$. Alerts are processed and then sent to the Prediction Engine by a mechanism known as the Alert Order Rectification (AOR) procedure. The alert buffer occupancy is constantly being monitored and the number of alerts in the buffer at any instance is accounted by the variable *buffer_occupancy*. The interval between successive AOR procedure executions is also tracked and time elapsed since the last alert AOR procedure is tracked by the variable *time_elapsed*. The AOR procedure is triggered when *buffer_occupancy* surpasses a configurable parameter called *alert sequence length* $\omega$ or *time_elapsed* exceeds another configurable parameter termed as *timeout* $\eta$. The AOR procedure triggered when *buffer_occupancy* exceeds $\omega$ follows a three step protocol. First, a forwarding window $F$ is formed that includes all alerts in $Y$ and all alerts received after waiting for $\mu_{max}$ period of time. Second, the alerts in $F$ are sorted with respect to $ts$. Finally, the first $\omega$ number of alerts in $F$ are sent to the Prediction Engine and are removed from $Y$. The $\mu_{max}$ wait time ensures that all alerts that are supposed to be part of $F$ according to their alert generation timestamp, but were not received due to network latencies are included in $F$. It is possible

that *buffer_occupancy* does not exceed $\omega$ for a long period of time. However, the alerts cannot remain in $Y$ indefinitely, and must be reported to the Prediction Engine for timely assessment of the attack. In that case, the AOR procedure is triggered when *time_elapsed* caps $\eta$, which also follows the three step protocol. The only difference is that all of the alerts in $Y$ are included in $F$, are sorted, and then sent to the Prediction Engine. As a result, a smaller sequence of alerts is sent to the Prediction Engine. The AOR procedure is articulated in Algorithm 2 and its time complexity is $O(n\,log(n))$, while $n$ here represents the number of alerts being sorted.

**Algorithm 2** Alert Order Rectification

**Input:** $Y = \{o_1, o_2, ..\}, \mu_{max}, \omega, \eta$

1: $buffer\_occupancy \longleftarrow no.\ of\ alerts\ in\ Y$
2: $time\_elapsed \longleftarrow time\ since\ last\ AOR\ proc.$
3: **while** $Y \neq \emptyset$ **do**
4:  **if** $(buffer\_occupancy \geq \omega)$ **then**
5:  $F \longleftarrow Y[o_1 : o_{t+\mu_{max}}]$
6:  $F.sort()$
7:  $F = F[0 : \omega]$
8:  $Y = Y - F$
9:  $sendToPredictionEngine(F)$
10:  $F \leftarrow \emptyset$
11:  **end if**
12:  **if** $(time\_elapsed == \eta)$ **then**
13:  $F \longleftarrow Y$
14:  $F.sort()$
15:  $Y = Y - F$
16:  $sendToPredictionEngine(F)$
17:  $F \leftarrow \emptyset$
18:  **end if**
19:  $update(buffer\_occupancy)$
20:  $update(time\_elapsed)$
21: **end while**

It is to be noted that for best results the value of $\mu_{max}$ needs to be as close as possible to $\Delta_{max}$. If we underestimate the value of $\mu_{max}$ then the rectification of alert reordering yields less precise results, and if the value of $\mu_{max}$ is overestimated then the original alert order can be restored by the ASM, but it will cause additional delay in sending alerts to the Prediction Engine. Also, the value set for $\omega$ controls the number of alerts to be processed

by the Prediction Engine at a time. Generally, longer alert sequences allow more accurate attack prediction but at the cost of delay in attack prediction decision. The value of $\eta$ should be set considering the alert arrival rate. It should not be too small to trigger the timeout driven forwarding process frequently and nor should be too large to introduce unnecessary delays in attack prediction. Fig 3.5 shows the state of $Y$ during the AOR procedure. The AOR procedure is triggered by buffer_occupancy exceeding $\omega$, and correspondingly, $F$ is constructed by including all alerts that arrive until the $\mu_{max}$ time as shown in Fig 3.5a. Subsequently, the first $\omega$ number of sorted alerts in $F$ are sent to the Prediction Engine and are discarded from $Y$ as illustrated in Fig. 3.5b. Similarly, the time_elaspsed triggered AOR procedure is shown in Figure 3.6. First, the formation of $F$ followed by sorting of the alerts is carried out. Afterwards, all alerts in $F$ are sent to the Prediction Engine before being discarded from the buffer.

### 3.2.4 Prediction Engine

Prediction engine performs the function of assessing the progression of an attack using the alerts forwarded by the ASM. PRISM has the flexibility to accommodate any time series data-based machine learning model, and in its current implementation the prediction engine is driven by HMM. Similar to TAS, prediction engine has offline and online modes of operation. In offline or training mode, prediction engine performs the operations of training data preprocessing and multi-stage attack learning. In online or prediction mode, attack stage prediction is carried out in realtime. The working of the internal modules of prediction engine is described as follows.

#### Training Data Preprocessing

The training of HMM utilizes IDS alert data that can be from IDS alerts of real attack instances, alerts from engineered attacks on a test network or synthetic alerts forged for training purposes. As the training data can be from multiple diverse sources, it is essential to formalize the preprocessing task. The training data preprocessing module accepts data in a CSV format then performs four step preprocessing operation. First, all rows with missing

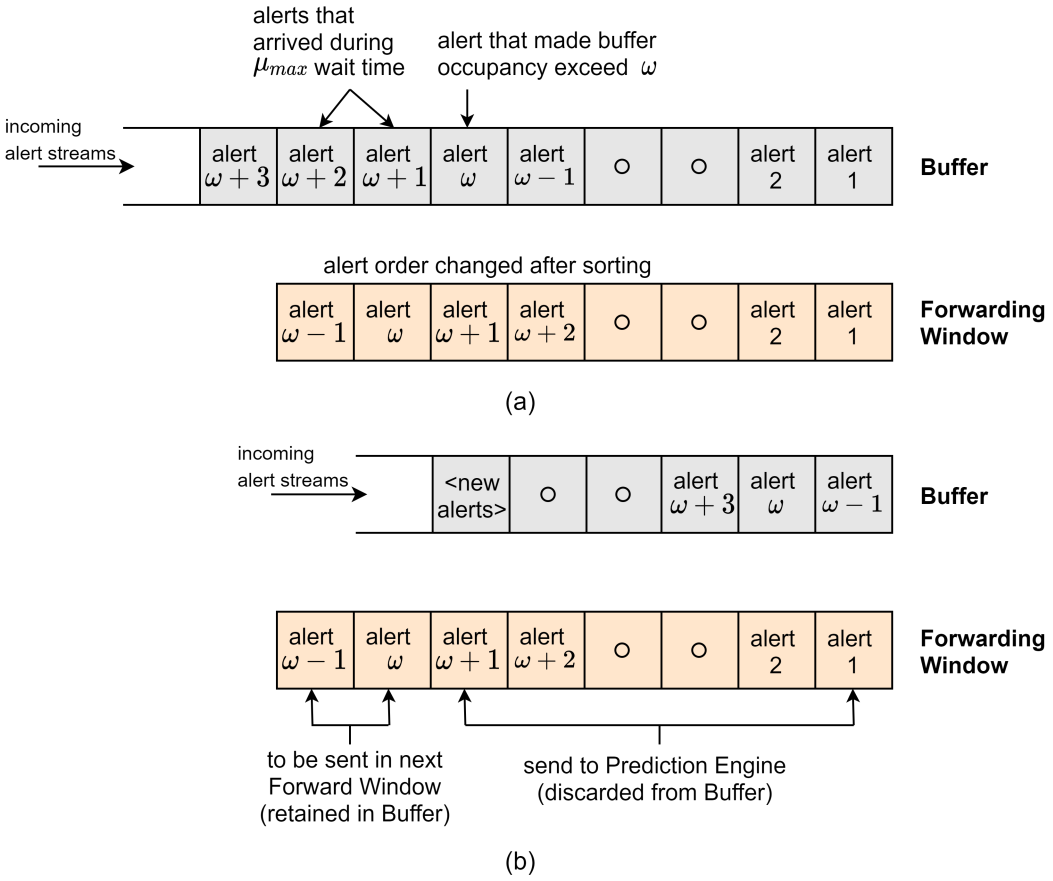


**Figure 3.5.** Evolution of buffer (Y) during the buffer_occupancy triggered AOR procedure. (a) Creation of forwarding window (F) and sorting of alerts. (b) Sending alerts to Prediction Engine.

values are removed. Second, the timestamp format of alerts is checked and is reformatted if it is incompatible with the data parsing function. Third, duplicate rows are discarded. Fourth, the alert data is rearranged in the ascending order with respect to the timestamps. Finally, the refined IDS alert data is stored in a CSV file ready to be used by the multi-stage attack learning module of the prediction engine.

### Multi-Stage Attack Learning

The multi-stage attack learning module analyzes the preprocessed IDS alert data files to train the HMM. For each multi-stage attack type in the training data, a separate model is trained forming an HMM attack profile bank that is shared with the attack stage prediction

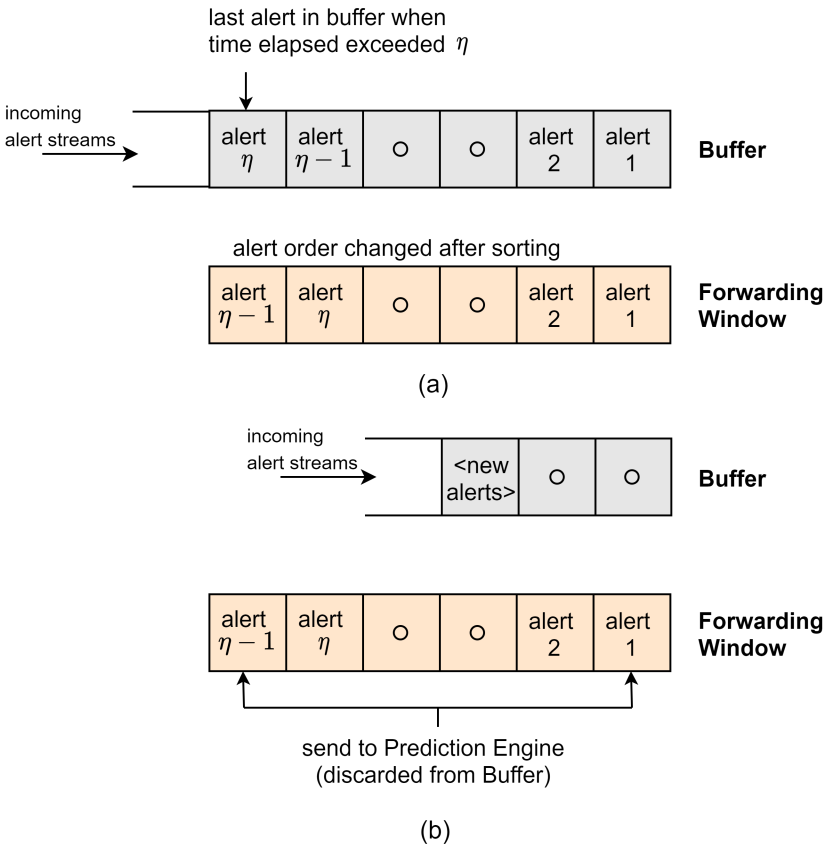


**Figure 3.6.** Evolution of buffer (Y) during the time_elapsed triggered AOR procedure. (a) Creation of forwarding window (F) and sorting of alerts. (b) Sending alerts to Prediction Engine.

module. PRISM uses the MITRE ATT&CK [80] adversary tactics and techniques knowledge base to contrive the state space of its prediction models that correspond to different attack progression stages. ATT&CK is a real-world observations-based threat information framework that describes the actions of adversaries in executing complex multi-stage attacks. These actions are represented by the techniques in ATT&CK framework that are organized into fourteen tactics. ATT&CK provides a detailed technical description, real-world usage examples with associated actors, mitigation and detection information for each technique. Tactics are the tactical objectives of the adversary that are achieved by employing the relevant techniques and tactics are arranged in the way attacker progresses through a multi-stage attack. The tactics in ATT&CK framework naturally become suitable candidates for the states in the multi-stage attack prediction models. However, having fourteen states not only

**Table 3.1.** Mapping of ATT&CK Tactics to Prediction States

| ATT&CK Tactics | Prediction States |
|---|---|
| i) Reconnaissance<br>ii) Resource development<br>iii) Initial access | 1) Initial access |
| iv) Execution | 2) Execution |
| v) Persistence<br>vi) Privilege escalation<br>vii) Defense evasion<br>viii) Credential access<br>ix) Discovery<br>x) Lateral movement<br>xi) Collection<br>xii) Command and control | 3) Foothold |
| xiii) Exfiltration<br>xiv) Impact | 4) Impact |

adds to computational complexity, but having all fourteen tactics involved in a single attack is highly unlikely. This necessitates the mapping of the fourteen tactics into a smaller and more generic state space. The prediction engine of PRISM maps the fourteen tactics into four states: initial access, execution, foothold, and impact. The mapping between ATT&CK tactics and the prediction states is shown in Table 3.1. Note that the terms attack stage and prediction state are used interchangeably in the paper. As discussed earlier, a supervised learning approach is used to train the HMM. The preprocessed IDS alert training data does not have labels indicating the classification of alert sequence into ATT&CK tactics and corresponding prediction states. Most of the mainstream threat detection frameworks have the feature to report alerts in compliance to the ATT&CK framework [81]–[83]. Since PRISM relies on the opensource snort IDS, a mechanism is required to associate snort alerts to the ATT&CK tactics. For this purpose, we have utilized the concepts of information retrieval to match the key words from snort alert messages to the technical description of the ATT&CK

techniques which is discussed in detail in Section 3.3 of the dissertation. Using the labelled IDS alert training data, $A$ and $B$ matrices are learned using Eqs. 3.7 and 3.8.

$$\begin{aligned} a_{\mathrm{ij}} &= \frac{\Gamma_{\mathrm{ij}}}{\sum_{r=1}^{N} \Gamma_{\mathrm{i}r}}, \qquad && \mathrm{i,j} \in [1, N] \\ s.t. \quad a_{\mathrm{ij}} &= 0, && \mathrm{j} > \mathrm{i} + 2 \\ & && \mathrm{j} < \mathrm{i} - 1 \end{aligned} \tag{3.7}$$

$$\begin{aligned} b_{\mathrm{i}}(l) &= \frac{\Upsilon_{\mathrm{i}}(l)}{\sum_{r=1}^{M} \Upsilon_{\mathrm{i}}(r)}, \qquad && \mathrm{i} \in [1, N] \\ & && l \in [1, M] \end{aligned} \tag{3.8}$$

Where, $\Gamma_{\mathrm{ij}}$ is the number of transitions from state $s_{\mathrm{i}}$ to $s_{\mathrm{j}}$ and $\Upsilon_{\mathrm{i}}(l)$ is the number of times observation $v_l$ appears in state $s_{\mathrm{i}}$. The learned HMM can be ergodic or of any shape based on the state transitions present in the training data. However, there are certain HMM types that model some applications more accurately, e.g., the left-right model is considered to have better performance in speech recognition [73]. Based on our experimentation, we have devised a semi-ergodic HMM for PRISM. The transition probabilities being learned in Eq. 3.7 are constrained by the two conditions that no transitions of more than two states are permitted and transitions from higher states to lower states are only possible among the adjacent states. These constraints improve the accuracy in detecting multi-stage attack progression. Algorithm 3 explains the multi-stage attack learning process where it receives the alert training data and produces the HMM attack profile bank $\Lambda$. It is to be noted that the models are designed to always begin from the first state thats why all models in $\Lambda$ are initialized with $\boldsymbol{\pi}_{\mathrm{i}} = [1\ 0\ 0\ 0]$.

**Algorithm 3** Multi-stage Attack Learning

**Input:** $Training\_Data = \{attack_1, attack_2, .., attack_k\}$

**Output:** $\Lambda = \{\lambda_1, \lambda_2, .., \lambda_k\}$

$\Lambda \longleftarrow$ *vector of* $k$ $\lambda$ *objects* (*init. with* $\pi_i$)
**for** $x \in [1, k]$ **do**
  /*Learning Transition Probabilities*/
  **for** i $\in [1, N]$ **do**
    **for** j $\in [1, N]$ **do**
      $\lambda_x.A[i][j] = \frac{\Gamma_{ij}}{\sum_{r=1}^{N} \Gamma_{ir}}$ /* from Eq. 3.7*/
      **if** (j $>$ i $+ 2$ || j $<$ i $- 1$) **then**
        $\lambda_x.A[i][j] = 0$
      **end if**
    **end for**
  **end for**
  /*Learning Emission Probabilities*/
  **for** i $\in [1, N]$ **do**
    **for** $l \in [1, M]$ **do**
      $\lambda_x.B[i][j] = \frac{\Upsilon_i(l)}{\sum_{r=1}^{M} \Upsilon_i(r)}$ /* from Eq. 3.8*/
    **end for**
  **end for**
**end for**

### Attack Recognition and Stage Prediction

The attack recognition and stage prediction module predicts the type of the multi-stage attack and its current stage for every alert sequence forwarded by the ASM in realtime. The attack stage prediction process has two parts, attack recognition and decoding. In attack recognition, the attack profile corresponding to the observed alert sequence is determined from the HMM attack profile bank. To find out the most likely attack profile, the likelihood of the alert observation sequence given the model, $P(O|\lambda)$, is computed for each HMM attack

profile using the forward algorithm. The HMM attack profile with the highest likelihood score is selected and is used to track the progress of the attack. Forward algorithm uses the forward variable $\zeta_t(\mathrm{i}) = P(o_1, o_2, .., o_T, q_t = s_\mathrm{i}|\lambda)$ which is the probability of the observation sequence $O = \{o_1, o_2, .., o_T\}$, and state at time $t$ being $s_\mathrm{i}$, given the model $\lambda$. Forward algorithm solves for $\zeta_t(\mathrm{i})$ inductively, using the initialization, induction and termination steps illustrated in Eqs. 3.9, 3.10 and 3.11, respectively.

$$\zeta_1(\mathrm{i}) = \boldsymbol{\pi}_\mathrm{i} b_\mathrm{i}(o_1), \qquad \mathrm{i} \in [1, N] \tag{3.9}$$

$$\zeta_{t+1}(\mathrm{j}) = \sum_{\mathrm{i}=1}^{N} \zeta_t(\mathrm{i}) a_{\mathrm{ij}} b_\mathrm{j}(o_{t+1}), \qquad \mathrm{j} \in [1, N] \quad t \in [1, T-1] \tag{3.10}$$

$$P(O|\lambda) = \sum_{\mathrm{i}=1}^{N} \zeta_T(\mathrm{i}) \tag{3.11}$$

After attack recognition, the process of decoding takes place that involves finding out the optimal state sequence corresponding to the forwarded alert sequence. To achieve this, the variable $\chi_t(\mathrm{i})$ is introduced that represents the probability of being in state $s_\mathrm{i}$ at time $t$ as expressed in Eq. 3.12. The most likely state $q_t$ at time $t$ can be solved independently using Eq. 3.13.

$$\chi_t(\mathrm{i}) = P(q_t = s_\mathrm{i}|O, \lambda) \tag{3.12}$$

$$q_t(\mathrm{i}) = \underset{\mathrm{i}=1,..,N}{\mathrm{argmax}} \chi_t(\mathrm{i}), \quad t \in [1, T] \tag{3.13}$$

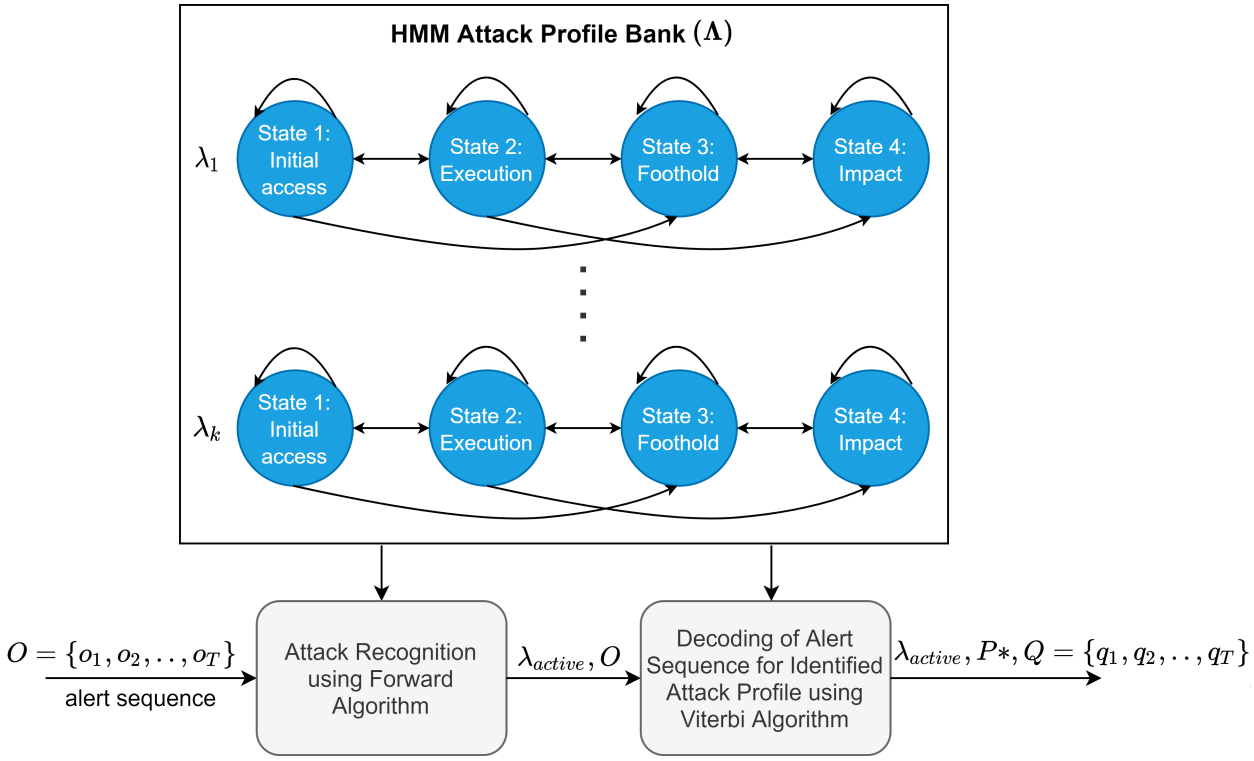


**Figure 3.7.** The process of attack recognition and decoding.

Viterbi algorithm takes a holistic approach by solving for the most likely state sequence $Q = \{q_1, q_2, .., q_T\}$ corresponding to the alert observation sequence $O = \{o_1, o_2, .., o_T\}$. The variable $\nu_t$, expressed in Eq. 3.14, navigates through the solution.

$$\nu_t(\mathrm{i}) = \max_{q_1, q_2, .., q_{t-1}} P(q_1, .., q_{t-1}, o_1, .., o_t, q_t = \mathrm{i}|\lambda) \tag{3.14}$$

Where, $\nu_t(\mathrm{i})$ is the maximum probability for a single state sequence accounting for $t$ observations and $s_\mathrm{i}$ being the last state in the state sequence. Using induction, Eq. 3.15 can be derived.

$$\nu_{t+1}(\mathrm{j}) = \max_{\mathrm{i}=1,..,N} \nu_t(\mathrm{i}) a_{\mathrm{ij}} b_\mathrm{j}(o_{t+1}) \tag{3.15}$$

To record the arguments that maximize Eq. 3.15, the array $\xi_t$ is introduced. Viterbi algorithm begins by initializing $\nu_t$ and $\xi_t$ for $t = 1$, as shown in Eqs. 3.16a and 3.16b.

$$\nu_1(\mathrm{i}) = \pi_\mathrm{i} b_\mathrm{i}(o_1), \qquad \mathrm{i} \in [1, N] \tag{3.16a}$$

$$\xi_1(\mathrm{i}) = 0, \qquad \mathrm{i} \in [1, N] \tag{3.16b}$$

The algorithm then recursively solves for each time step $t = 2$ *to* $T$ and each state $\mathrm{j} = 1$ *to* $N$, using Eqs. 3.17a and 3.17b.

$$\nu_t(\mathrm{j}) = \max_{\mathrm{i}=1,..,N} \nu_{t-1}(\mathrm{i}) a_\mathrm{ij} b_\mathrm{j}(o_t), \quad t \in [2, T], \mathrm{j} \in [1, N] \tag{3.17a}$$

$$\xi_t(\mathrm{j}) = \operatorname*{argmax}_{\mathrm{i}=1,..,N} \nu_{t-1}(\mathrm{i}) a_\mathrm{ij}, \quad t \in [2, T], \mathrm{j} \in [1, N] \tag{3.17b}$$

Finally, the probability of the best score sequence is determined using Eq. 3.18a and the best path state is discovered in Eq. 3.18b.

$$P^* = \max_{\mathrm{i}=1,..,N} \nu_T(\mathrm{i}) \tag{3.18a}$$

$$q_T^* = \operatorname*{argmax}_{\mathrm{i}=1,..,N} \nu_T(\mathrm{i}) \tag{3.18b}$$

Eq. 3.19 finds out the best state sequence starting at the best path state and backtracking in time by following $\xi_t$.

$$q_t^* = \xi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, .., 1 \tag{3.19}$$

Further details about the forward and Viterbi algorithms can be found in [73], [84]. Algorithm 4 explains the functioning of attack recognition and stage prediction module systematically. The time complexity of Algorithm 4 is $O(N^2T)$ which is the time complexity of both forward and Viterbi algorithms. Fig 3.7 depicts the use HMM attack profile bank in the attack recognition and decoding tasks.

### Adversarial Machine Learning for Hidden Markov Model-based Threat Prediction

The HMM architecture constructed in the multi-stage attack learning process is capable to detect a wide variety of multi-stage attack scenarios as illustrated in Section 3.4 of the dissertation. However, the architecture is prone to adversarial examples that can obfuscate the prediction system. One such scenario is the activity of an adept attacker who has knowledge about the working of the Prediction Engine of PRISM. In order to deceive the prediction process, the attacker intentionally performs the activities that generate alerts for an earlier stage of the multi-stage attack while actually being at an advanced stage.

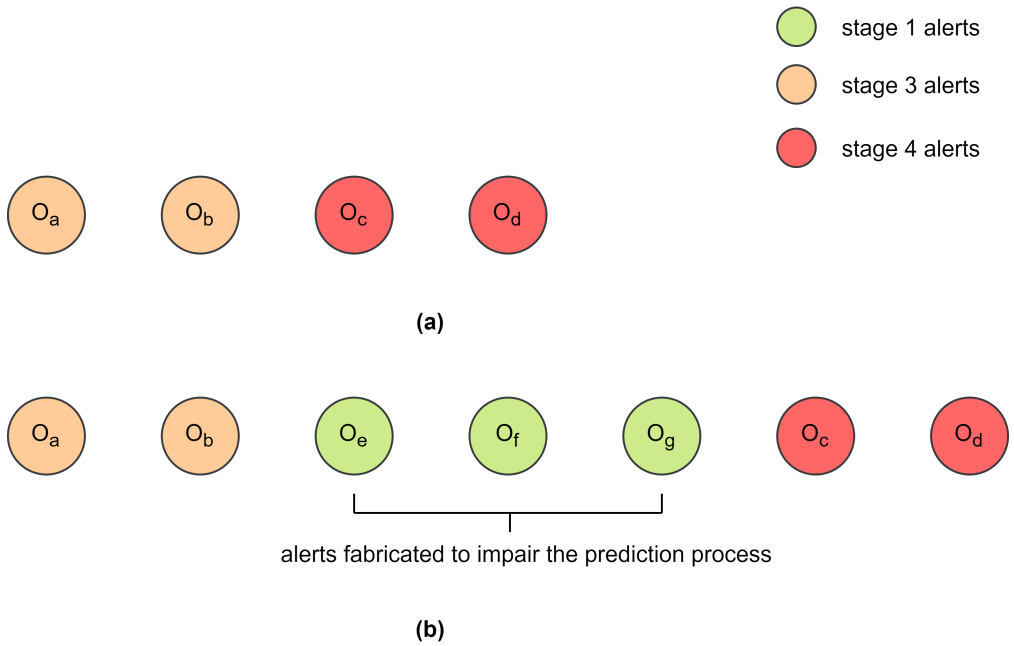


**Figure 3.8.** Adversarial machine learning scenario for HMM-based prediction system. (a) Expected alert sequence corresponding to attack progression. (b) Modified alert sequence on account of the attacker's evasion strategy.

Consider the scenario where the attack is at stage 3 and natural progression of the attack is supposed to be towards stage 4 as shown in Figure 3.8a. Now, to deceive the prediction model, the attacker deliberately repeats some of his actions that generate alerts corresponding to stage 1 of the attack as shown in Figure 3.8b. If the alerts generated for the purpose of deceiving the prediction model belong to some other attack type then the Prediction Engine of PRISM will initiate a new attack instance. However, if the alerts are from the same attack type then Prediction Engine is prone to end up in an error state, leaving the attacker to progress to advanced stages without being detected. Such adversarial examples against HMM-based intrusion detection has been studied previously. In [63], HMM architectures have been proposed that have the ability to differentiate between instances of multiple attacks of the same type with interleaved alerts. The architectures propose the inclusion of an extra observation symbol and prediction state to address the issue of encountering unexpected alerts at a particular stage in addition to preprocessing the alert stream using an alert feature-based demultiplexer. Such architectures can be incorporated in PRISM, but currently they are not employed as in our threat model we have assumed that the attacker has no information about the inner workings of the prediction mechanism.

### 3.2.5 Security Analyzer

Security Analyzer realizes the cyber situational awareness capability of PRISM by the computation of important security metrics and their visualization in realtime for effective decision making in response to the attacks. The input to the Security Analyzer is prediction output of the Prediction Engine and unmediated alerts from the surveillance zones. By unmediated it is meant that the alerts are not subjected to the AOR procedure in the ASM. The alert and prediction output data is processed in realtime to extract relevant information for metrics computation. The information of interest in the alert data is the alert id, alert generation timestamp and id of the device for which the alert has been generated. From the prediction output, information regarding attack type, attack stage and the probability of being at different attack stages corresponding to an alert is obtained. The metric computation process is alert-driven (event-driven), and the metrics are computed with each

**Algorithm 4** Attack Recognition and Stage Prediction

**Input:** $O = \{o_1, o_2, .., o_T\},\ \Lambda = \{\lambda_1, \lambda_2, .., \lambda_k\}$
**Output:** $\lambda_{active},\ P*,\ Q = \{q_1, q_2, .., q_T\}$

```
/* Attack Recognition by Forward Algorithm */
max_likelihood = 0
for λ ∈ Λ do
    likelihood = P(O|λ) /* from Eq. 3.11 */
    if (likelihood > max_likelihood) then
        max_likelihood = likelihood
        λ_active = λ
    end if
end for
/* Decoding using Viterbi Algorithm */
ν_t(i) = max_{q_1,..,q_{t-1}} P(q_1, .., q_{t-1}, o_1, .., o_t, q_t = i|λ_active)
P* = max_{i=1,..,N} ν_T(i) /* from Eq. 3.18a */
Q = argmax_{i=1,..,N} ν_T(i) /* from Eq. 3.18b */
for t = T − 1 : 1 do
    Q = Q ∪ ξ_{t+1}(q*_{t+1})
end for
Q = Q.reverse()
```

reported alert at time $t$. To get a holistic picture of the damage spread, widely used system availability metric is utilized. Additionally, we propose two metrics: threat perceptivity and system degradability that are designed to enhance the cyber situational awareness capabilities specifically for the case of availability attacks. Formally, the three metrics are defined as follows.

**System availability:** The percentage of devices working at their routine operational capacity with respect to the total number of devices in the system at a certain time $t$. It is expressed in Eq. 3.20.

$$system\ availability(t) = \frac{no.\ of\ available\ devices\ at\ t}{total\ no.\ of\ devices} * 100 \tag{3.20}$$

**Threat perceptivity:** The measure to quantify attack progression and associated risk to the system. Mathematically, it is expressed in Eq. 3.21, where $\chi_t(\mathrm{i})$ is the probability of being at state $s_\mathrm{i}$ for an alert received at time $t$, $\varepsilon_\mathrm{i}$ is the numerical value corresponding to

the risk associated with state $s_i$, with higher values to be set for advanced stages. The value of thereat perceptivity is between 0 and 1 and higher value means the ongoing attack is in advanced stages with the system facing elevated risk.

$$\textit{threat perceptivity}(t) = \frac{\sum_{i=1}^{N} \chi_t(i)\varepsilon_i}{max(\varepsilon_i)} \tag{3.21}$$

**System degradability:** The impact of attack risk on the system operability quantified through the fraction of compromised devices in the system. Eq. 3.22a specifies the fraction of compromised devices at time $t$, and system degradability at time $t$ is manifested in Eq. 3.22b. Like threat preceptivity, the value of system degradability is also between 0 and 1, with higher values implying increased attack progression intensity.

$$\theta(t) \;=\; \frac{\textit{no. of compromised devices at } t}{\textit{total no. of devices}} \tag{3.22a}$$

$$\textit{system degradability}(t) = \theta(t)\ \textit{threat perceptivity}(t) \tag{3.22b}$$

System availability, threat perceptivity and system degradability is computed for each reported alert and the Security Analyzer plots the metrics in realtime for the convenience of the security operations team to gain valuable insights about the ongoing attack. Visualization of the metrics helps to assess the situation better and consequently a more informed response can be deployed. Additionally, all of the metric computations are constantly being stored in a report file that can be used by an automated response system. The visualization functionality of the Security Analyzer is discussed further in Section 3.4 of the dissertation.

## 3.3 Experimental Setup

To evaluate the performance of PRISM, comprehensive experimentation is conducted on the ISCX-2012 dataset. The dataset contains a wide variety of multi-stage attacks embedded in seven days of network activity. In this dissertation we will be concentrating on the infiltration and HTTP DoS attack types. Fig. 3.9 shows the attack progression in the

infiltration attack scenario and Fig. 3.10 shows the attack progression in the HTTP DoS attack scenario. The attacks are launched on a network of hundreds of workstations and the captured traffic is stored in the packet capture (pcap) files. The network activity data of a day is stored in a separate file and includes up to 10 million packets. The dataset also incorporates labelled flow-data files providing information whether a flow is malicious or benign. All constituent systems of PRISM are implemented in Python 3.7 and are compatible with the specifications of the dataset. The experiments are conducted on a workstation with 8 cores and 16 GB of memory. Fig. 3.11 shows the conceptual deployment of PRISM on ISCX-2012 network architecture by realizing each Local Area Network (LAN) as a surveillance zone. The specific details of how different modules of the constituent systems of PRISM implement the distributed security monitoring and integrated security analysis functionality using the dataset resources is discussed as follows.

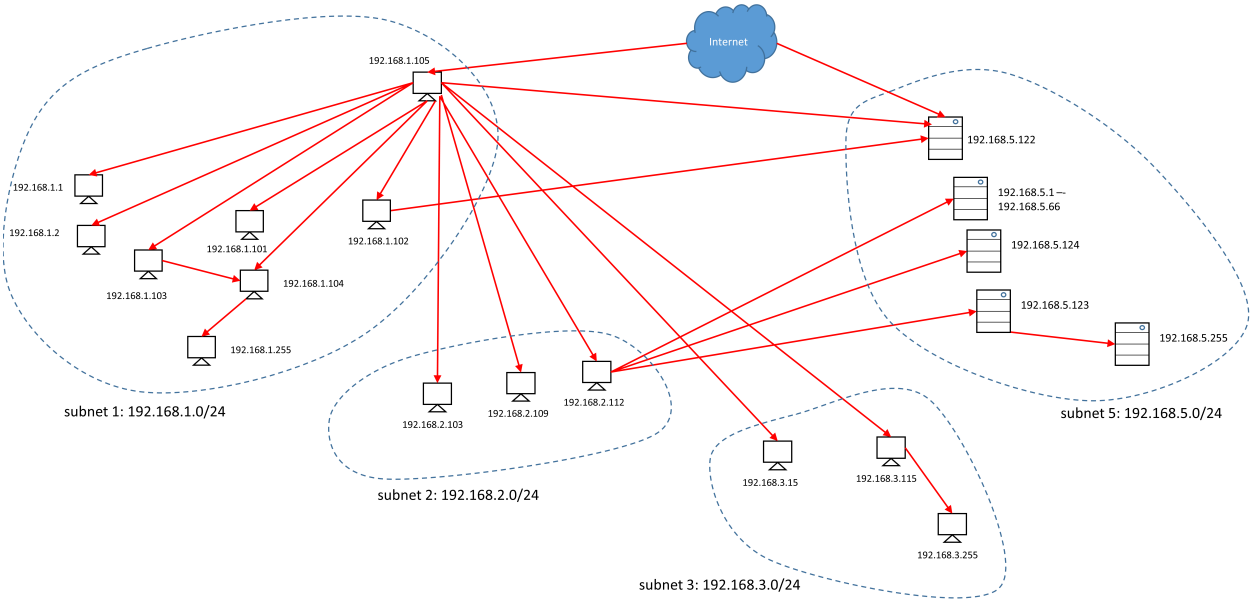


**Figure 3.9.** Attack progression in the infiltration attack scenario of ISCX-2012 dataset.

### 3.3.1 Distributed Security Monitoring

TAS and Zonal IDS implement the distributed security monitoring functionality of PRISM. As mentioned in Section 3.2, TAS performs the operations of network reachability graph

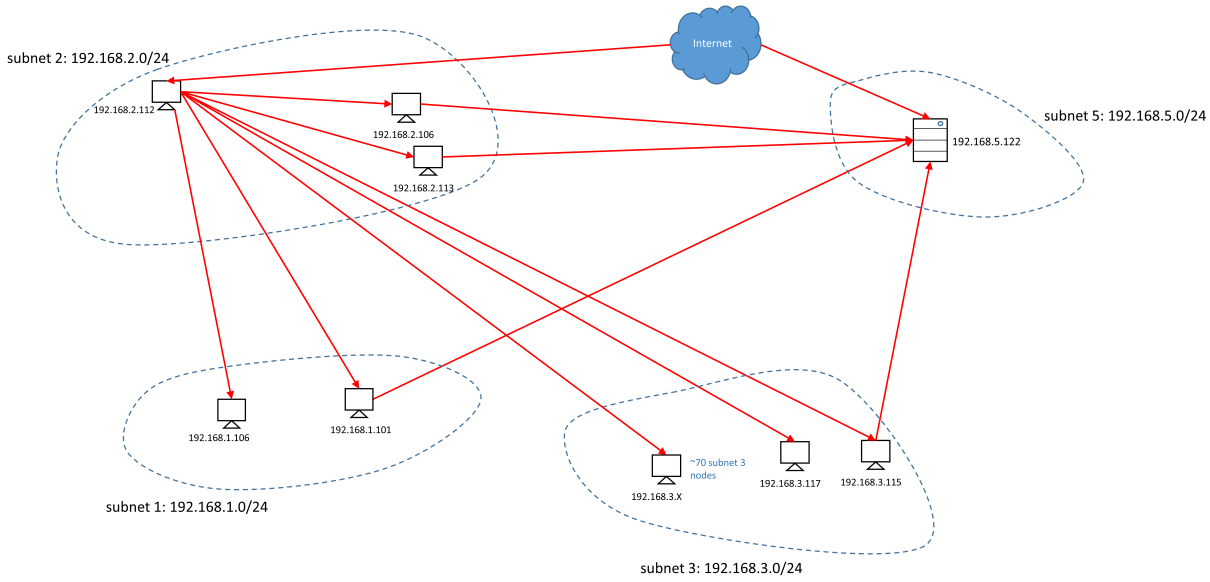


**Figure 3.10.** Attack progression in the HTTP DoS attack scenario of ISCX-2012 dataset.

generation, vulnerability assessment, vulnerability ranks and sampling probabilities computation, and realtime probabilistic sampling. The network reachability graph generation is carried out by determining the communication between all network devices on different port numbers using the normal network activity data in the dataset. For vulnerability assessment, the vulnerability score of each device is estimated as we cannot deploy vulnerability scanners on the ISCX-2012 network nor vulnerability information of the network devices is provided in the dataset. To estimate the vulnerability scores of devices, we use the CVSS base score computation methodology by treating network devices as vulnerabilities to be exploited. The CVSS base score is determined using five exploitability metrics: attack vector, attack complexity, privileges required, user interaction and scope, along with three impact metrics: confidentiality, integrity and availability. The attack vector is determined using network reachability graph with the assumption that all devices that can communicate to a device can also compromise it. Therefore, the part of the network from where a device could potentially be targeted can be determined. Since we are modeling multi-stage attacks, the attack complexity is set to *high* for each device. Privileges required is set as *low* for regular hosts and *high* for servers. User interaction is set as *required* and scope is set as *changed* for each device. For all three impact metrics, the value set to *low* for regular hosts and *high* for

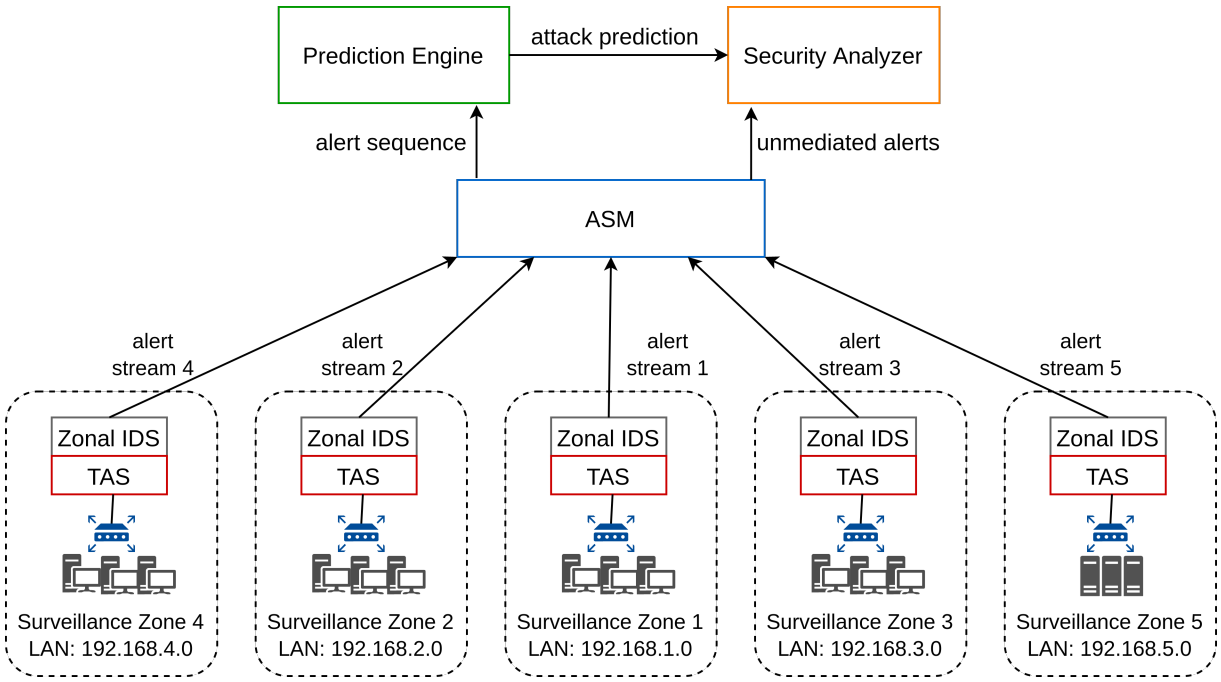


**Figure 3.11.** PRISM deployment on the ISCX-2012 network.

servers. Table 3.2 summarizes the values adjusted for CVSS base score metrics. Once the network reachability graph is generated and the vulnerability assessment process is complete, the vulnerability ranks and sampling probabilities of all devices in the network are calculated using Algorithm 1. The network traffic incorporated in pcap files is then distributed between the surveillance zones using the IP addresses of the devices. In each surveillance zone, the packets in the zonal pcap file are sampled using Eq. 3.6. The sampled zonal pcap file is then forwarded to Snort IDS that generates an alert file using Snort v3.0 community rules. To engineer the distributed alert reporting process, the alert file in each surveillance zone is appended with a transmission delay value between $\Delta_{min}$ and $\Delta_{max}$, which are configurable parameters in our experiments, and for each surveillance zone, the transmission delay values are randomly chosen within the range of these parameters. The alerts from each surveillance zone are then delivered to the input buffer of the ASM according to the alert generation timestamp added with the transmission delay.

### 3.3.2 Integrated Security Analysis

The integrated security analysis functionality of PRISM is implemented by ASM, Prediction Engine and Security Analyzer. Initially, the multi-stage attack learning operation of the Prediction Engine is performed to construct the HMM attack profile bank. The pcap files for different attack types in the dataset are used to generate the alert files using Snort IDS. The four-step training data preprocessing is carried out and then the alert labelling corresponding to ATT&CK tactics is accomplished. To label an alert, keywords are extracted from the alert information and are matched to the technical description of ATT&CK techniques using the information retrieval method of Term Frequency - Inverse Document Frequency (TF-IDF). The TF-IDF scores of each keyword are calculated for all ATT&CK techniques and the technique that has the highest average TF-IDF score of the alert message keywords is identified. The ATT&CK tactic that pairs to the identified ATT&CK technique is selected, and prediction state corresponding to the selected tactic, as described in Table 3.1, is chosen as the label of the alert. Some ATT&CK techniques are part of multiple tactics, therefore, labelling of alerts requires establishment of context. The context is established by using the state corresponding to the last labelled alert. For multiple candidate tactics picked up for an alert, those tactics are selected that maps to a higher prediction state, otherwise the alert is labelled with the state of the last labeled alert. The semantics behind this rule is that as the attacker is executing a multi-stage attack, it is counter intuitive for the attacker to move in the backwards direction. If there are several alerts pointing to the previous prediction states then that will be considered as a new multi-stage attack. Such attack scenarios are investigated in [63]. In rare cases, the average TF-IDF score of alert message keywords is not significantly different for different ATT&CK techniques due to limitation of expressiveness in Snort alert information. To handle such cases we label the alert with the label of the previous alert, i.e., we make the assumption that this alert is not causing any attack stage transition. Once the alert files are labelled, Algorithm 3 trains the HMM attack profiles using 30% of the alert data for each attack type in the ISCX-2012 dataset.

**Table 3.2.** CVSS Metrics Values for Vulnerability Assessment

| Metric Name | Metric Value |
|---|---|
| Attack Vector | Determined by NRG |
| Attack Complexity | High |
| Privileges Required | Regular hosts: low, Servers: high |
| User Interaction | Required |
| Scope | Changed |
| Confidentiality | Regular hosts: low, Servers: high |
| Integrity | Regular hosts: low, Servers: high |
| Availability | Regular hosts: low, Servers: high |

Alerts from different surveillance zones are gathered in the input buffer of the ASM, from where they are sent to the Prediction Engine and Security Analyzer. The alerts being sent to the Prediction Engine are first sequenced and then forwarded using the AOR procedure articulated in Algorithm 2. The value of the parameter $\mu_{max}$ is configured to be less than the value set for $\Delta_{max}$ in the experiments to test the robustness of alert stream management process. The parameter $\omega$ is selected according to the maximum length of alert sequence intended to be processed by the Prediction Engine. The value of the $\eta$ parameter is fixed to be 500 ms in all experiments. Prediction Engine upon reception of an alert sequence first finds out which attack is active and then predicts the stage of the attack using Algorithm 4. Security Analyzer receives unmediated alerts from the ASM and attack prediction information from the Prediction Engine to compute and visualize three metrics: system availability, threat perceptivity and system degradability for each incoming alert.

## 3.4 Performance Evaluation

We have conducted several experiments to evaluate the performance of PRISM in terms of processing efficiency and prediction efficacy. The experiments are designed to demonstrate the individual performance of the constituent systems of PRISM and their influence on the overall system operability.

### 3.4.1 Effect of Threat-Aware Sampling and Distributed Traffic Processing

The most computation intensive phase in the intrusion detection exercise is the processing of the network traffic. The vast magnitude of traffic generated by various devices in the enterprise networks make it even more challenging especially when the requirement is to detect malicious traffic in realtime. To cope up with this challenge, a simple yet expensive solution is commonly employed, that is, adding more computation power. PRISM on the other hand provides an effective solution that leverages its threat-aware sampling and distributed traffic monitoring structure to process traffic in realtime with limited computation resources.

#### Post Sampling Information Retention

In the first experiment, the performance of TAS and d-TAS is compared to common network traffic sampling schemes of smart sampling and random sampling. In smart sampling flows are sampled according to their size with large flows having more probability of being selected [85]. The comparison between the sampling schemes is made using a measure that determines how much information is lost in the sampling process. In intrusion detection applications, a malicious flow is the information that is desired to be retained where a normal flow is considered as noise. We introduce the metric Malicious-flow Loss Rate (MLR) that determines the amount of malicious flows not retained after sampling as expressed in Eq. 3.23. Sampling ratio $\sigma$ is the control parameter in sampling related experiments, which is the number of flows selected over the total number of flows as manifested in Eq. 3.24.

$$MLR = \frac{number\ of\ malicious\ flows\ not\ selected}{total\ number\ of\ malicious\ flows} \tag{3.23}$$

$$\sigma = \frac{number\ of\ sampled\ flows}{total\ number\ of\ flows} \tag{3.24}$$

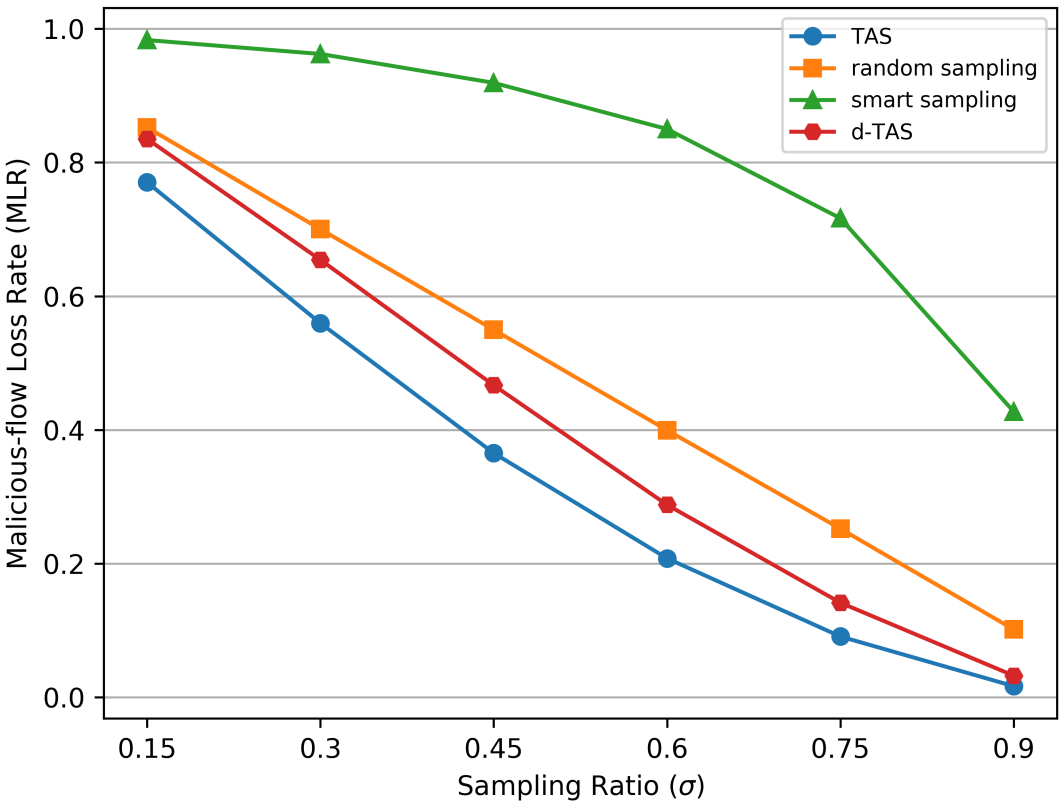


**Figure 3.12.** TAS performance in comparison to common network traffic sampling schemes for infiltration attack data.

The MLR of TAS, d-TAS, random sampling and smart sampling corresponding to different values of $\sigma$ for infiltration attack scenario in the ISCX-2012 dataset is shown in Fig. 3.12. It can be seen that the MLR of TAS and d-TAS is significantly better than that of random sampling and smart sampling with the widest difference observed for $\sigma = 0.45$ and $\sigma = 0.6$. Note that the MLR values of all sampling schemes for $\sigma = 0.15$ and $\sigma = 0.90$ do not differ much as compared to the other values of $\sigma$. This is because if the sampling rate is low then only a few flows are being selected by all sampling schemes that leaves less room for TAS to fully utilize its adversary behavior model-based sampling approach. For high sampling rate, since only a few flows are dropped, therefore, random sampling and smart sampling are also able retain many malicious flows.

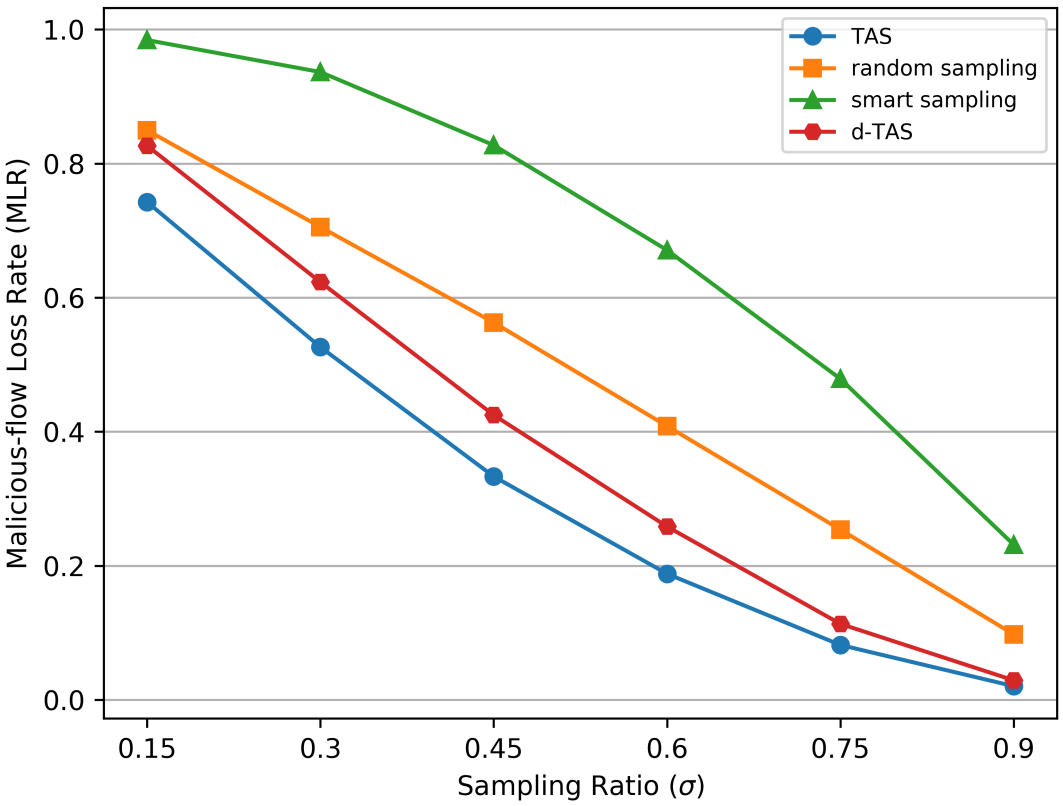


**Figure 3.13.** TAS performance in comparison to common network traffic sampling schemes for HTTP DoS attack data.

For HTTP DoS attack, The MLR values of TAS, d-TAS, random sampling and smart sampling corresponding to different values of $\sigma$ are shown in Fig. 3.13. The MLR in the case of smart sampling is better in HTTP DoS attack scenario as compared to the infiltration attack. This is due to the fact that in infiltration attacks, the flow size is generally limited to few hundred bytes, however, in HTTP DoS attack, the size of the flows is significantly larger. As mentioned earlier that the smart sampling scheme favors flows with large volumes, so the large sized flows captured by the smart sampling scheme includes higher number of malicious flows. Therefore, the MLR for smart sampling is comparatively better in the HTTP DoS attack scenario. The performance of the other three sampling schemes is similar to that in the case of infiltration attack with TAS and d-TAS performing significantly better than random sampling and smart sampling techniques.

### Network Traffic Processing Overhead

In the second experiment, we have investigated the effect of TAS and distributed architecture on the network traffic processing capabilities of PRISM in comparison to Snort being employed in a centralized setting.

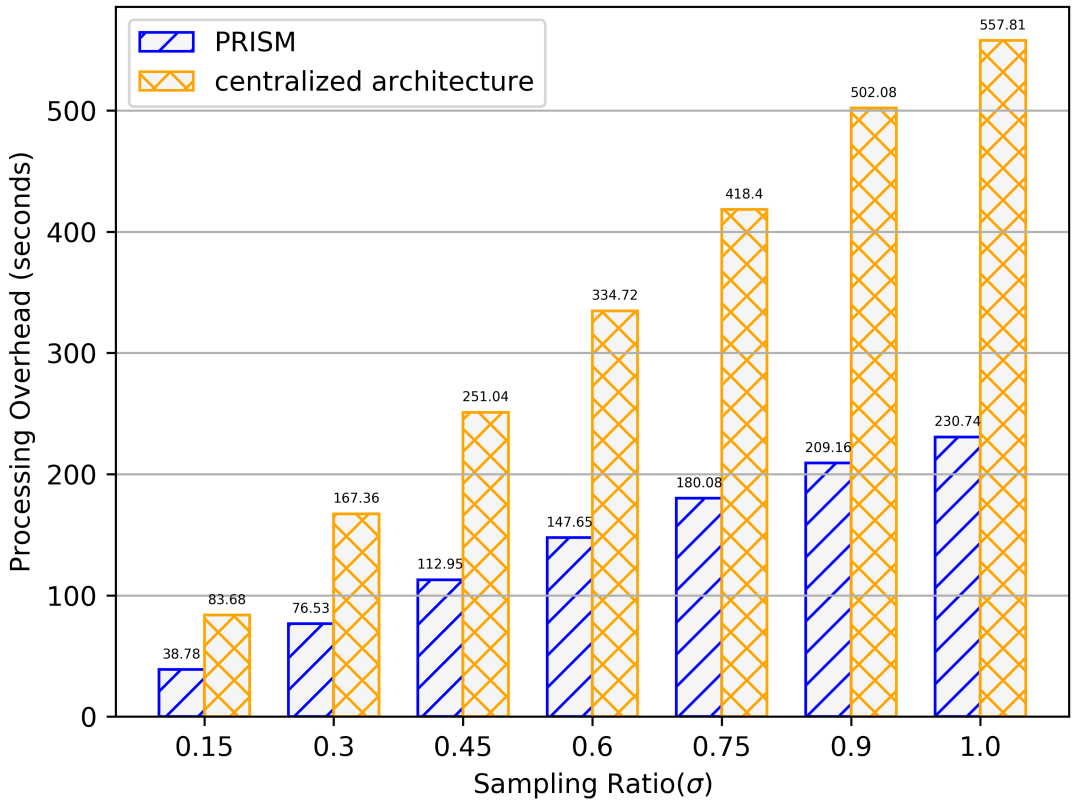


**Figure 3.14.** Processing overhead of centralized intrusion detection and distributed security monitoring of PRISM corresponding to different sampling ratios for infiltration attack data.

Fig. 3.14 shows the processing overhead of Snort for around 5.8 million packets of infiltration attack data. As expected, the performance of PRISM's distributed architecture is significantly better than a centralized intrusion detection architecture for different values of $\sigma$. It has been determined in our other experiments discussed ahead that the prediction capabilities of PRISM are acceptable with even $\sigma = 0.3$. Comparing the processing over-

head of PRISM's distributed traffic processing architecture with $\sigma = 0.3$ to the processing overhead of non-distributed and no sampling system ($\sigma = 1$) as an equivalent to a standard IDS (like Snort) shows that PRISM makes the intrusion detection process 7.3x faster in the case of infiltration attack scenario.

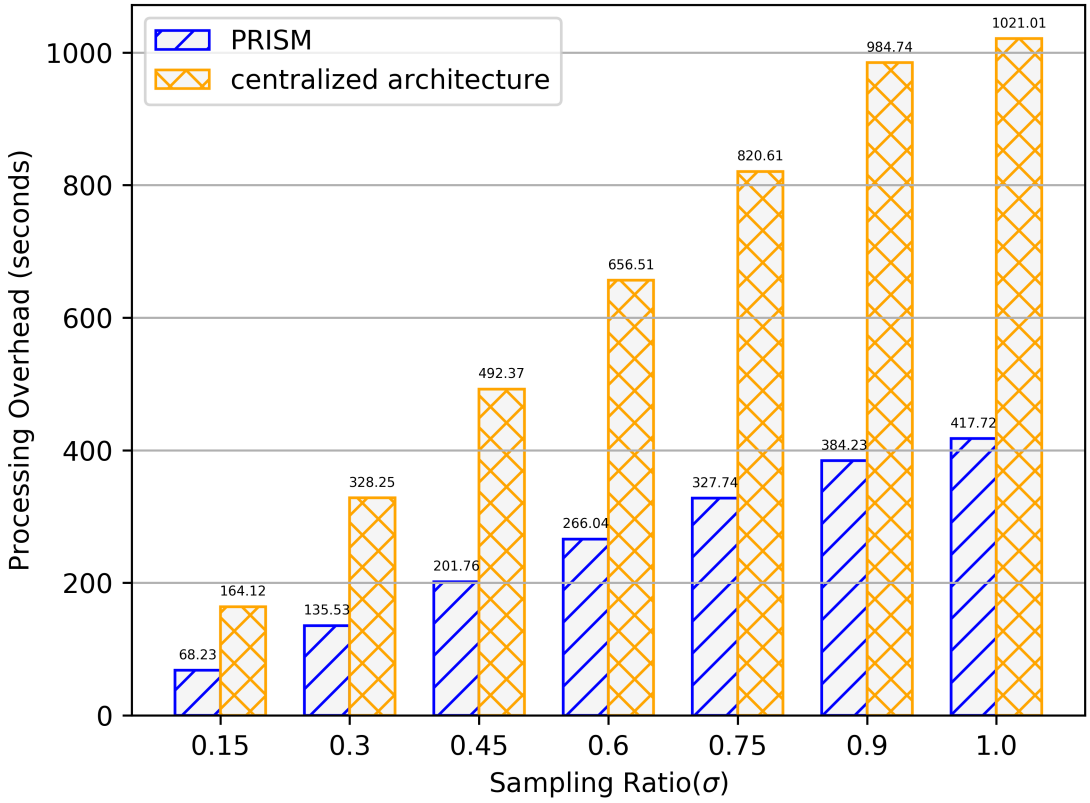


**Figure 3.15.** Processing overhead of centralized intrusion detection and distributed security monitoring of PRISM corresponding to different sampling ratios for HTTP DoS attack data.

The processing overhead of snort for around 9.6 million packets of HTTP DoS attack data is shown in Fig. 3.15. The processing overhead values in the case of HTTP DoS attack are similar to that of infiltration attack scenario with the overhead of PRISM being almost one-third of a standard centralized IDS architecture. Similar to infiltration attack scenario, the prediction capabilities of PRISM are acceptable for $\sigma = 0.3$ and comparing the processing overhead of PRISM's distributed architecture setting with $\sigma = 0.3$ to the

processing overhead of non-distributed and no sampling system ($\sigma = 1$) as an equivalent to a standard centralized IDS shows that the intrusion detection process with PRISM is 7.5x faster in the HTTP DoS attack scenario.

### 3.4.2 Attack Prediction Performance and Utility of Alert Stream Management

In this section we present extensive evaluation of the prediction capabilities of PRISM through rigorous experimentation and all of the results presented in this section correspond to the models trained using 30% of the alert data from the infiltration attack and HTTP DoS attack scenarios in the ISCX-2012 dataset. First, we discuss the performance of PRISM in identifying correct attack stages for different sampling ratios. Second, the prediction recall corresponding to different alert sequence lengths is illustrated. Third, we evaluate the effect of distributed alert reporting process on the prediction performance by demonstrating the prediction recall for different alert delay settings. Finally, we evaluate the rapidness in identifying attack stages by studying the impact of different delay configurations on early detection of an attack stage and how PRISM mitigates the issue by utilizing its alert stream management capabilities.

#### Attack Stage Prediction for Different Sampling Ratios

We have performed several experiments to determine the prediction efficacy of PRISM by varying the sampling ratios $\sigma$ of TAS while the alert sequence length is fixed to 10, that is, $\omega = 10$. The experiments are designed to showcase how different sampling ratios effect the prediction capabilities of PRISM and provide insights about the potential use of TAS with a particular sampling ratio in a given threat prediction scenario. There are two types of representations used to depict the prediction output of PRISM and how it fairs against the actual labels in our testing data. The first one shows the predicted attack stage for each alert in the training data and also shows the correct actual stage for the corresponding alert. The second one illustrates the the number of alerts correctly predicted by PRISM. Additionally, for the alerts that were incorrectly predicted to be from other stages, information regarding

how many alerts were incorrectly predicted and to what stage is presented in the form of a confusion matrix.

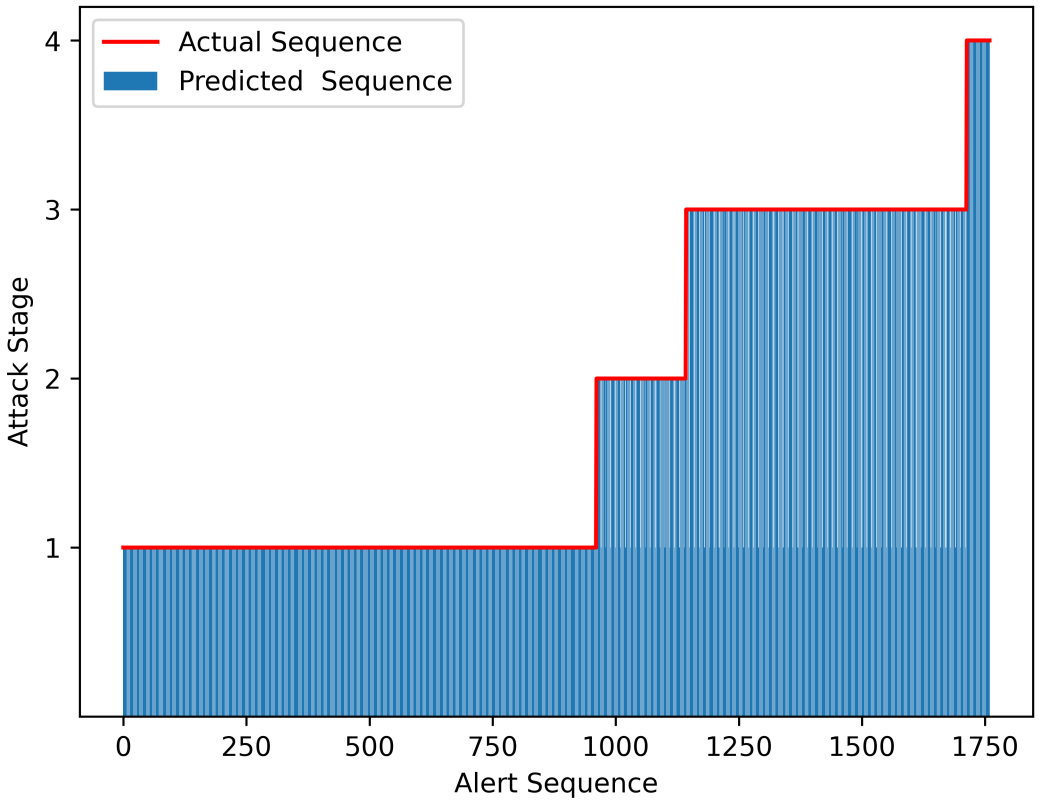


**Figure 3.16.** Attack stage prediction for infiltration attack with $\sigma = 0.15$.

The prediction output of PRISM corresponding to $\sigma = 0.15$ for infiltration attack scenario is presented in Fig. 3.17. It can be seen that PRISM is able to predict all stages correctly even when the number of alerts corresponding to a particular attack stage are lower. The number of alerts for stage 2 and stage 4 are lower as compared to the number of alerts in stage 1 and stage 3. This makes it a bit harder for the prediction model to determine the stages with fewer number of alerts, however, it can be seen that PRISM is able to determine stages 2 and 4 accurately. Moreover, it can be seen that the length of attack stage 4 quite small as compared to the other attack stages in the infiltration attack scenario and this fact will play an important role in the experimental results being presented later in this section.

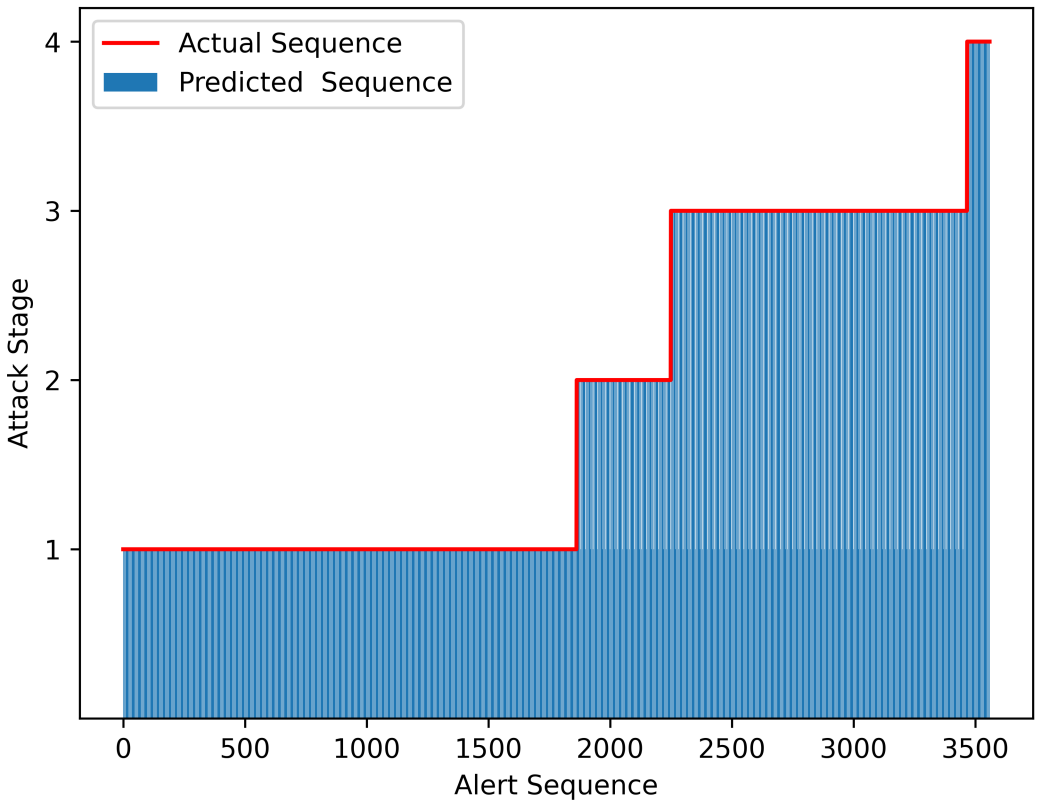


**Figure 3.17.** Attack stage prediction for infiltration attack with $\sigma = 0.30$.

In Fig. 3.17, the prediction output of PRISM with $\sigma = 0.3$ is illustrated for the infiltration attack scenario. The prediction output is similar to that of the case with $\sigma = 0.15$ as PRISM is able to identify all of the stages correctly, however, the length of the attack stages corresponding to the number alerts has increased with the increase in sampling ratio. Moreover, the relative lengths of the different attack stages are also similar to that of the experiment with $\sigma = 0.15$. Therefore, the length of attack stage 4 is still relatively smaller than the attack stages 1, 2 and 3. By length of an attack stage we mean the number of alerts corresponding to that attack stage, therefore, smaller length attack stages have fewer number of alerts representing them.

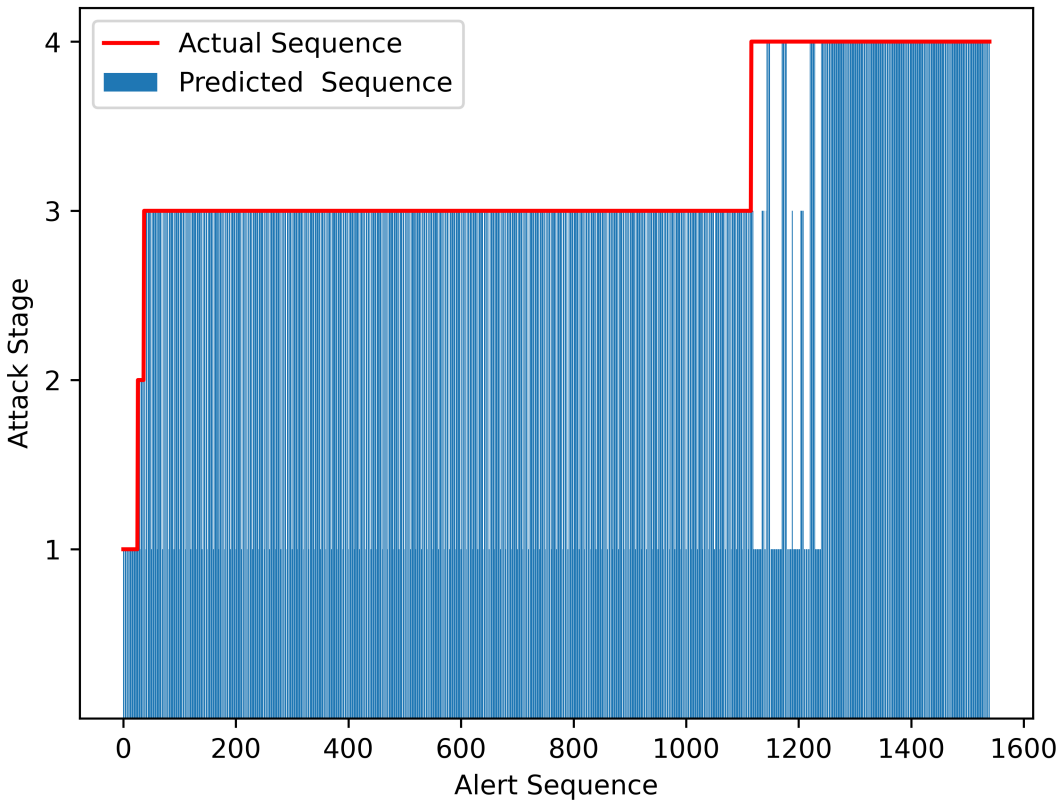


**Figure 3.18.** Attack stage prediction for HTTP DoS attack with $\sigma = 0.15$.

In the case of HTTP DoS attack, the lengths of stages 1 and 2 are significantly smaller as compared to stages 3 and 4 for $\sigma = 0.15$ as shown in Fig. 3.18. The prediction output for stage 3 is similar to the actual sequence with some discrepancies. In stage 4 of the HTTP DoS attack, it can be observed that initially a lot of alerts are incorrectly determined to be of stages 2 and 3 but as more alerts from stage 4 are processed by the prediction model, the prediction output becomes accurate. Furthermore, the length of the attack stage 2 is quite small as compared to the lengths of the attack stages 3 and 4. This observation will play an important role in the experiments presented later in this section.

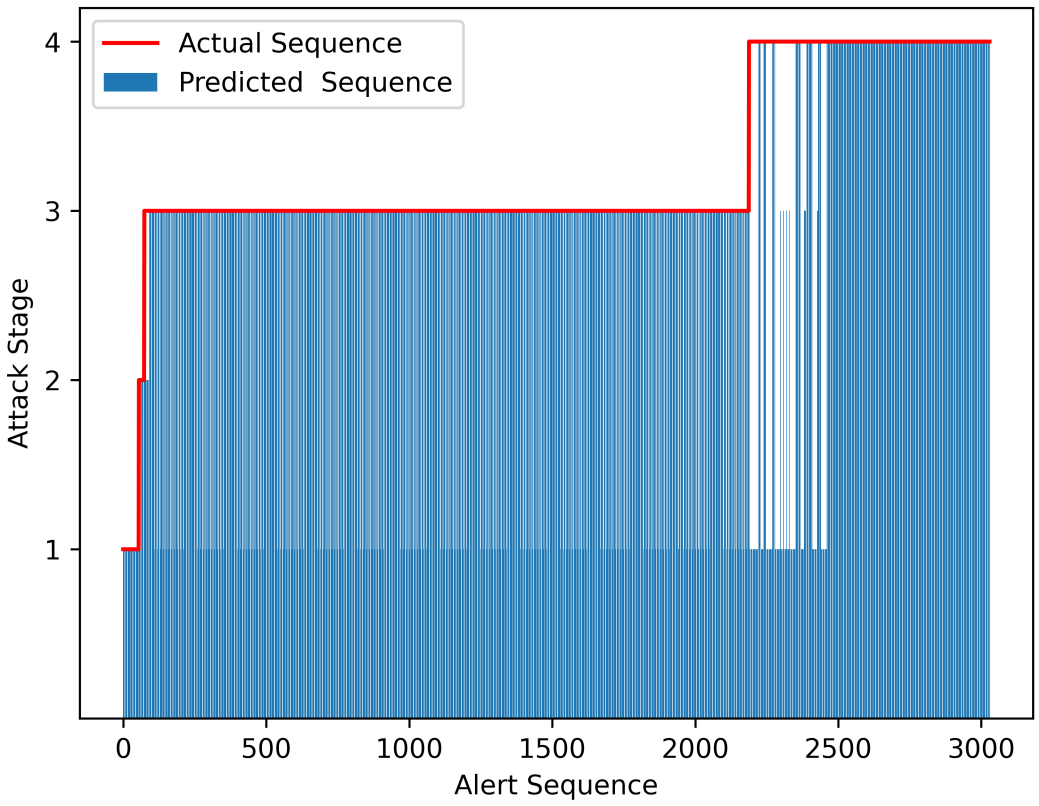


**Figure 3.19.** Attack stage prediction for HTTP DoS attack with $\sigma = 0.30$.

The prediction output of PRISM in the HTTP DoS attack scenario for $\sigma = 0.30$ is shown in Fig. 3.19. It can be seen that PRISM is able to determine stage 1 quite accurately, but for stage 2, some of the alerts from stage 3 are predicted to be of stage 2 in contrast to prediction output with $\sigma = 0.15$. In some cases the compatibility of the alert sequence length and the attack stage length is better, therefore, even with a lower sampling ratio of $\sigma = 0.15$, the prediction output is more accurate with $\omega = 10$ as compared to the prediction output for $\sigma = 0.30$ with $\omega = 10$. Though with the increase in the sampling ratio, the number of alerts per stage have increased proportionally but the relative lengths of the attack stages are similar to the prediction output for $\sigma = 0.15$. Therefore, the length of attack stage 2 is still quite small as compared to the lengths of the attack stages 3 and 4.

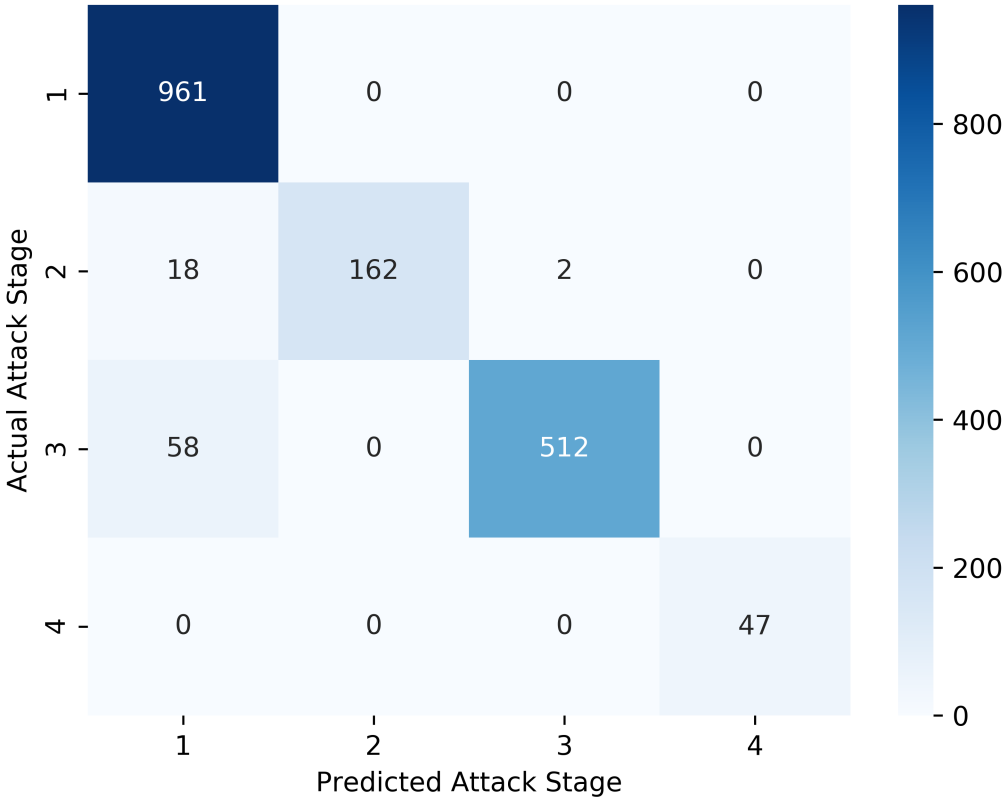


**Figure 3.20.** Confusion matrix representation of prediction output for infiltration attack with $\sigma = 0.15$.

The prediction output of PRISM with more details in the infiltration attack scenario with $\sigma = 0.15$ is shown in Fig. 3.20. It is illustrated that all alerts from attack stage 1 have been correctly identified. For attack stage 2 most incorrectly predicted alerts have been detected of being from attack stage 1 while some incorrectly predicted alerts have been detected of being from attack stage 3. For attack stage 3 all incorrectly predicted alerts are identified of being from stage 1. For attack stage 4, all alerts have been correctly identified. The confusion matrix helps to unravel the tendency of the prediction system to incorrectly predict a particular attack stage to other attack stages. For example, the Prediction Engine of PRISM has the tendency to incorrectly predict stages 2 and 3 as stage 1 in the infiltration attack scenario.

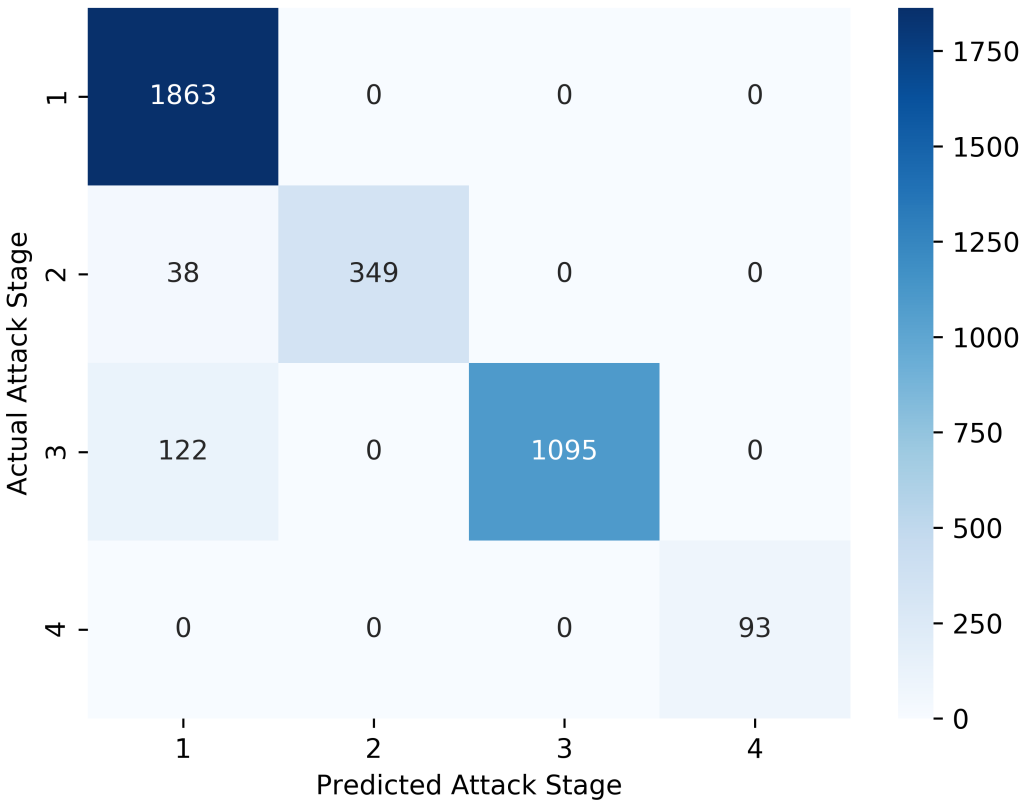


**Figure 3.21.** Confusion matrix representation of prediction output for infiltration attack with $\sigma = 0.30$.

Fig. 3.21 shows the classification details of prediction output in the form of confusion matrix for infiltration attack scenario when $\sigma = 0.30$. It can be seen that the overall prediction performance is fairly decent for the aforementioned training data and sampling ratio setting with $\omega = 10$. The stages 1 and 4 of the attack are predicted without any error while stages 2 and 3 are incorrectly predicted as stage 1 for some instances. Unlike in the case when $\sigma = 0.15$, none of the alerts from attack stage 2 are incorrectly predicted of being from attack stage 3. Moreover, the relative number of correctly predicted alerts to the incorrectly predicted alerts for $\sigma = 0.30$ in a particular attack stage are similar to that in the case of $\sigma = 0.15$.

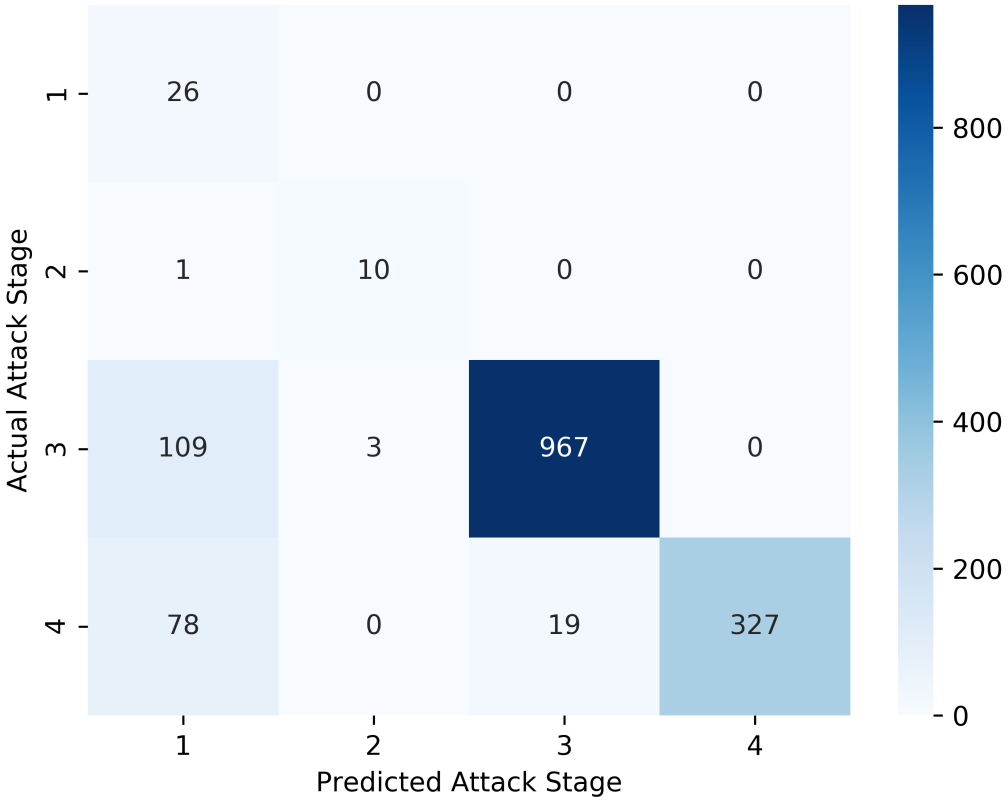


**Figure 3.22.** Confusion matrix representation of prediction output for HTTP DoS attack with $\sigma = 0.15$.

For HTTP DoS attack, the prediction output details for $\sigma = 0.15$ in the form of a confusion matrix is shown in Fig. 3.22. Most of the incorrect predictions are for attack stages 3 and 4. The prediction capabilities of PRISM are primarily dependant on the quality of data used in training the models. However, given a trained model, the runtime performance of the Prediction Engine is influenced by $\omega$, and generally longer length alert sequences are predicted by the model with more accuracy. But formation of long sequences needs increased waiting times to gather alerts, and in a realtime environment, the requirement is to process the alerts as quickly as possible for a timely prediction decision. Therefore, there is a trade off between prediction accuracy and the capability of being able to get the prediction output rapidly.

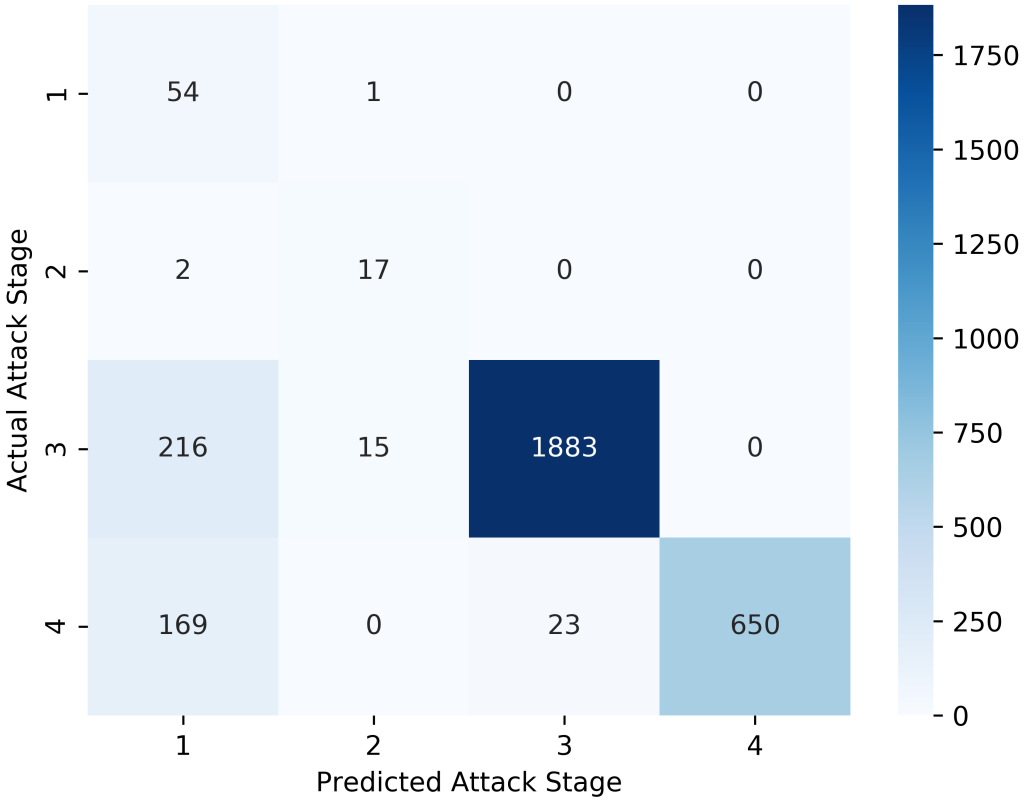


**Figure 3.23.** Confusion matrix representation of prediction output for HTTP DoS attack with $\sigma = 0.30$.

The prediction output details for $\sigma = 0.30$ in the HTTP DoS attack scenario is shown in Fig. 3.22. The prediction output is similar to that of the case of $\sigma = 0.15$, with the difference only being in the number of alerts being greater in the case of $\sigma = 0.30$. For attack stage 1, only one instance is being incorrectly predicted to be from attack stage 2, and for attack stage 2, two alerts are incorrectly predicted of being from attack stage 1. For attack stage 3, most of the incorrectly identified alerts are predicted to be from attack stage 1 and some instances are incorrectly predicted of being from attack stage 2. For attack stage 4, majority of the incorrectly predicted alerts are wrongly identified as being from stage 1 while some of the incorrectly predicted alerts are detected to be from stage 3.

### Prediction Recall for Different Alert Sequence Lengths

We have conducted experiments to reveal the effect of different alert sequence lengths on the prediction efficacy and to measure the prediction correctness, we have used recall that quantifies how well the model performs in identifying the actual attack stage in a given configuration.

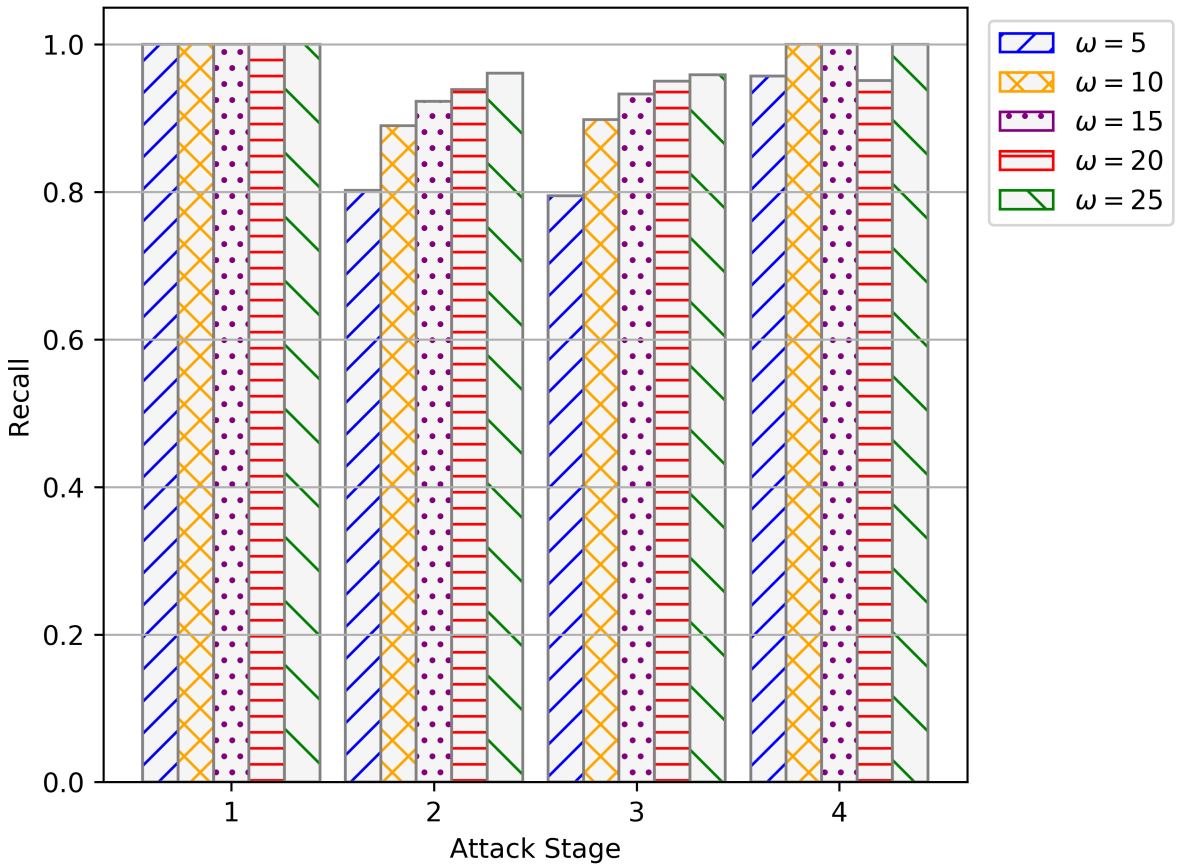


**Figure 3.24.** Prediction recall for different alert sequence lengths ($\omega$) for infiltration attack with $\sigma = 0.15$.

The prediction recall values in the infiltration attack scenario with $\sigma = 0.15$ for different values of $\omega$ is shown in Fig. 3.24. It can be seen that the prediction recall for all values of $\omega$ is ideal for attack stage 1. For attack stages 2 and 3, the prediction recall increases as the value of $\omega$ increases. For attack stage 4, the recall corresponding to $w = 5$ and $\omega = 20$ is less than 1 unlike the recall for $\omega = 10$, $\omega = 15$ and $\omega = 25$.

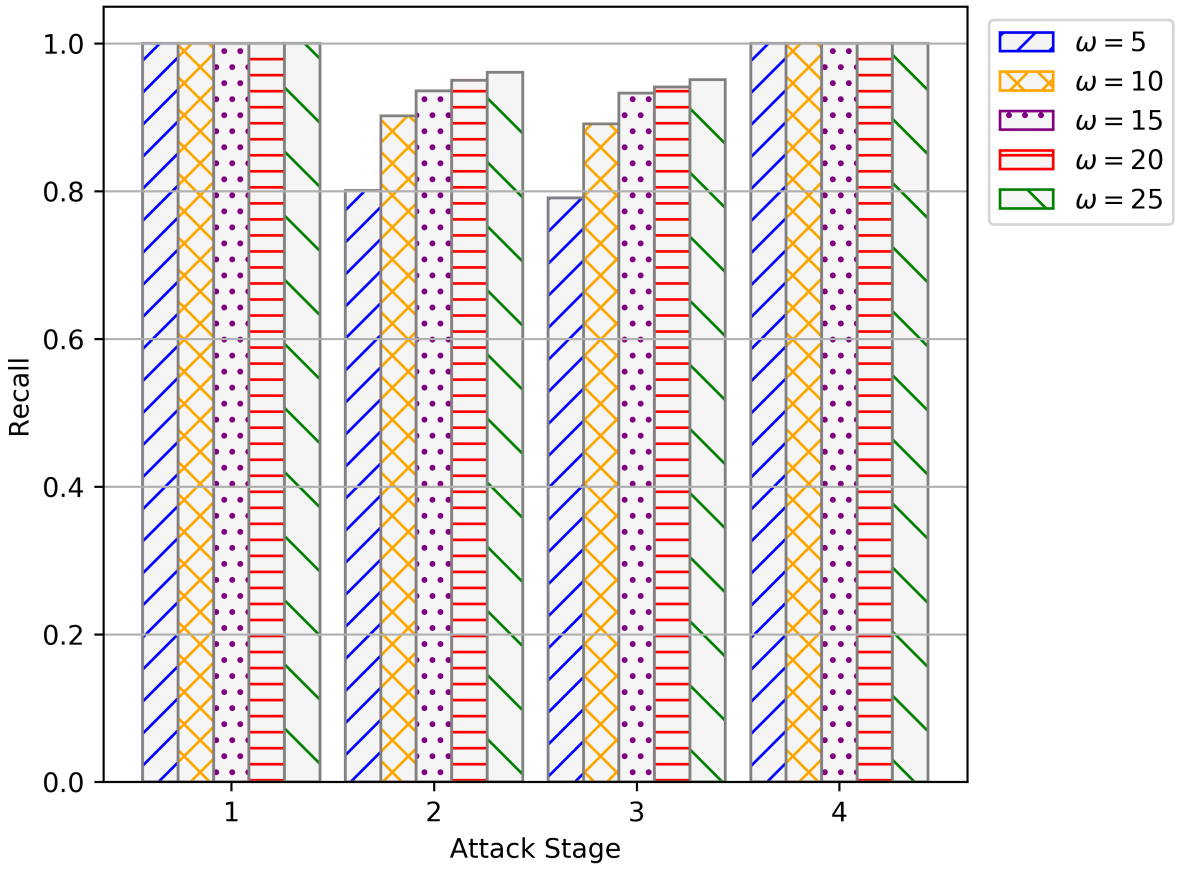


**Figure 3.25.** Prediction recall for different alert sequence lengths ($\omega$) for infiltration attack with $\sigma = 0.30$.

Fig. 3.25 shows the prediction recall values for infiltration attack scenario with $\sigma = 0.30$ corresponding to different values of $\omega$. It can be seen that the prediction recall for stages 1 and 4 is ideal for all values of $\omega$, but for stages 2 and 3, as the value of $\omega$ increases, the prediction recall also increases. It can be observed that for attack stages 2 and 3, the increase in the prediction recall when the values of $\omega$ changes from 5 to 10 is significant as compared to the increase in the prediction recall when the value of $\omega$ changes from 10 to 15 and so on. This is due to the fact that with the increase in $\omega$, that is, the length of the alert sequence to be processed by the prediction model, the prediction performance of the model improves, but that improvement becomes less for higher values of alert sequence lengths as the prediction model gets almost the same kind of information from the observation sequences after a certain a length of the alert sequence.

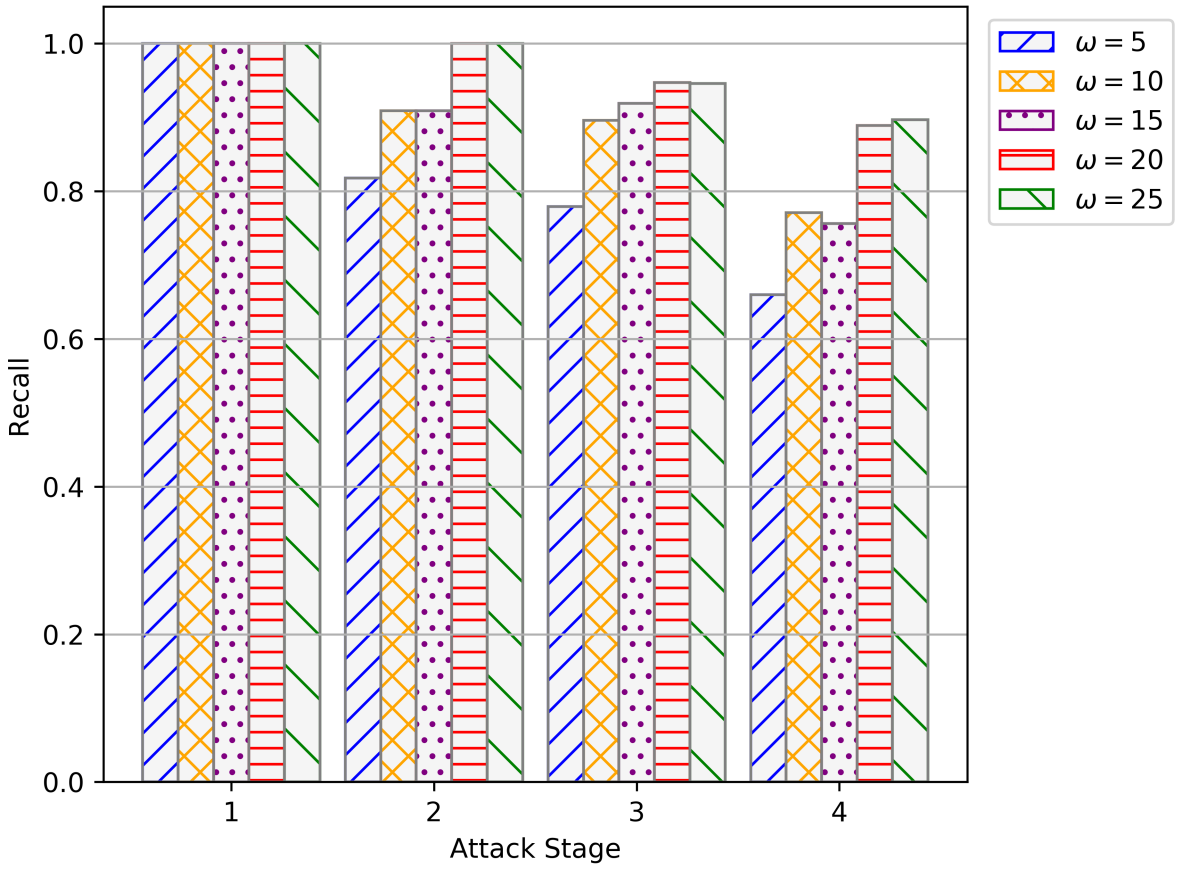


**Figure 3.26.** Prediction recall for different alert sequence lengths ($\omega$) for HTTP DoS attack with $\sigma = 0.15$.

The prediction recall corresponding to $\sigma = 0.15$ in the HTTP DoS attack scenario for different values of $\omega$ is shown in Fig. 3.26. It can be observed that the prediction recall for all values of $\omega$ in stage 1 of the attack is ideal. For stage 2 and stage 3 of the attack, the prediction recall increases as the values of $\omega$ increase. The prediction recall corresponding to stage 4 of the attack has relatively low recall as compared to other stages of the attack, however, the trend is the same that with the increasing $\omega$ values the prediction recall increases. However, for attack stage 4, the prediction recall value of $\omega = 10$ is higher than the prediction recall value for $\omega = 15$. This is due to the fact that for two different $\omega$ values the number of alert sequences formed from the total number of alerts present in the data and the makeup of those sequences is different. In rare cases, it has been observed that alert sequences formed with a lower value of $\omega$ have a better makeup of alerts then the alert sequences formed with a

higher value $\omega$. Therefore, the performance of the prediction mechanism is somewhat better for the shorter alert sequence length than an alert sequence length that is slightly longer.

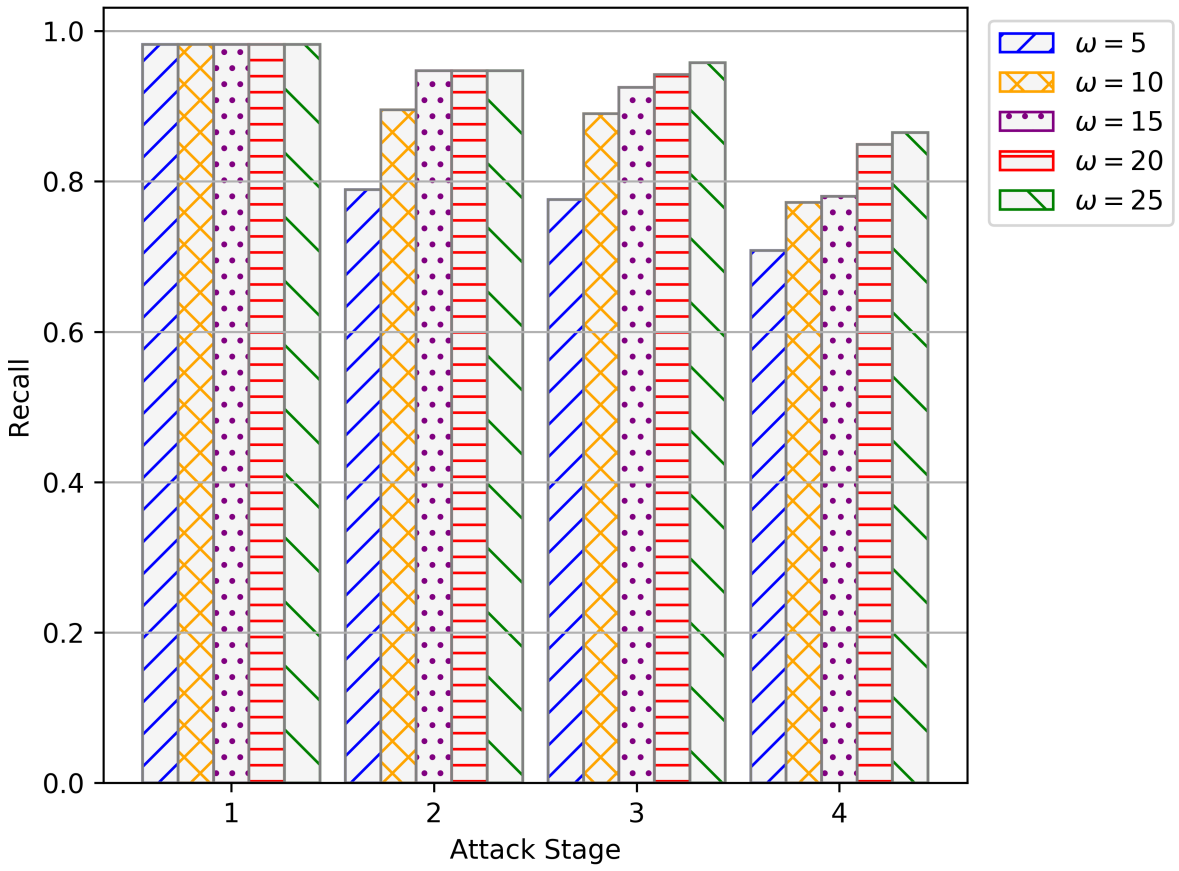


**Figure 3.27.** Prediction recall for different alert sequence lengths ($\omega$) for HTTP DoS attack with $\sigma = 0.30$.

Fig. 3.27 presents the prediction recall for HTTP Dos attack and it can be seen that the prediction recall improves with the increase in $\omega$ values for stage 2, 3 and 4. If Figs. 3.21 and 3.23 are observed together with Figs. 3.25 and 3.27, then it can be realised that lower values of $\omega$ exacerbate the error in prediction, especially for stages that have a significant number alert misclassifications. Moreover, the increase in prediction recall when the value of $\omega$ changes from 5 to 10 is more significant as compared to the increase in prediction recall when $\omega$ changes from 10 to higher values can also be observed here.

**Table 3.3.** Distributed Alert Reporting Parameters

| **Delay Configuration** | $\boldsymbol{\Delta_{min} - \Delta_{max}}$ **(ms)** | $\boldsymbol{\mu^{10}_{max}}$ **(ms)** | $\boldsymbol{\mu^{30}_{max}}$ **(ms)** | $\boldsymbol{\mu^{50}_{max}}$ **(ms)** |
|---|---|---|---|---|
| low | 50 - 100 | 90 | 70 | 50 |
| medium | 100 - 250 | 225 | 175 | 125 |
| high | 150 - 400 | 360 | 280 | 200 |

### Prediction Recall for Different Alert Delay Configurations

To ascertain the effect of alert stream management on the prediction performance we have engineered distributed alert reporting mechanism as discussed in Section 3.3 of the dissertation. The experimentation has been carried out with $\omega = 10$ using the three delay configurations: low, medium and high. Each delay configuration corresponds to a range of delay values, where for a given alert and a delay configuration, a delay value randomly chosen between $\Delta_{min}$ and $\Delta_{max}$ is added to the alert generation timestamp. The value of $\mu_{max}$ is set to be 10%, 30% and 50% lower than the value of $\Delta_{max}$ for low, medium and high delay configurations, respectively, and are represented by $\mu^{10}_{max}$, $\mu^{30}_{max}$ and $\mu^{50}_{max}$. Table 3.3 shows different ranges of the three delay configurations used in our experiments and corresponding values of $\mu_{max}$ are also depicted. The larger the range from which the alert delay value is chosen to be appended to the alert, the more randomness or distortion is observed to the sequence of alerts reported at the ASM from different Zonal IDSs. As we will show in the results of the distributed alert reporting experiments, with increasing distortion in the sequence of alerts, the prediction performance of the HMM-based Prediction Engine of PRISM is deteriorated. In fact, this phenomenon is true for all time series data-based prediction models as retention of the order of data is critical for their proper functioning. In order to test the robustness of the proposed AOR procedure employed by the ASM, we have used three different estimates of the maximum alert reporting delay for each delay configuration. These different values of our estimate of the maximum alert reporting delay evaluate the performance of PRISM stream management under practical consideration of having an imperfect maximum alert reporting delay estimate and how the increase in error of our estimate impact the threat prediction capability of PRISM.

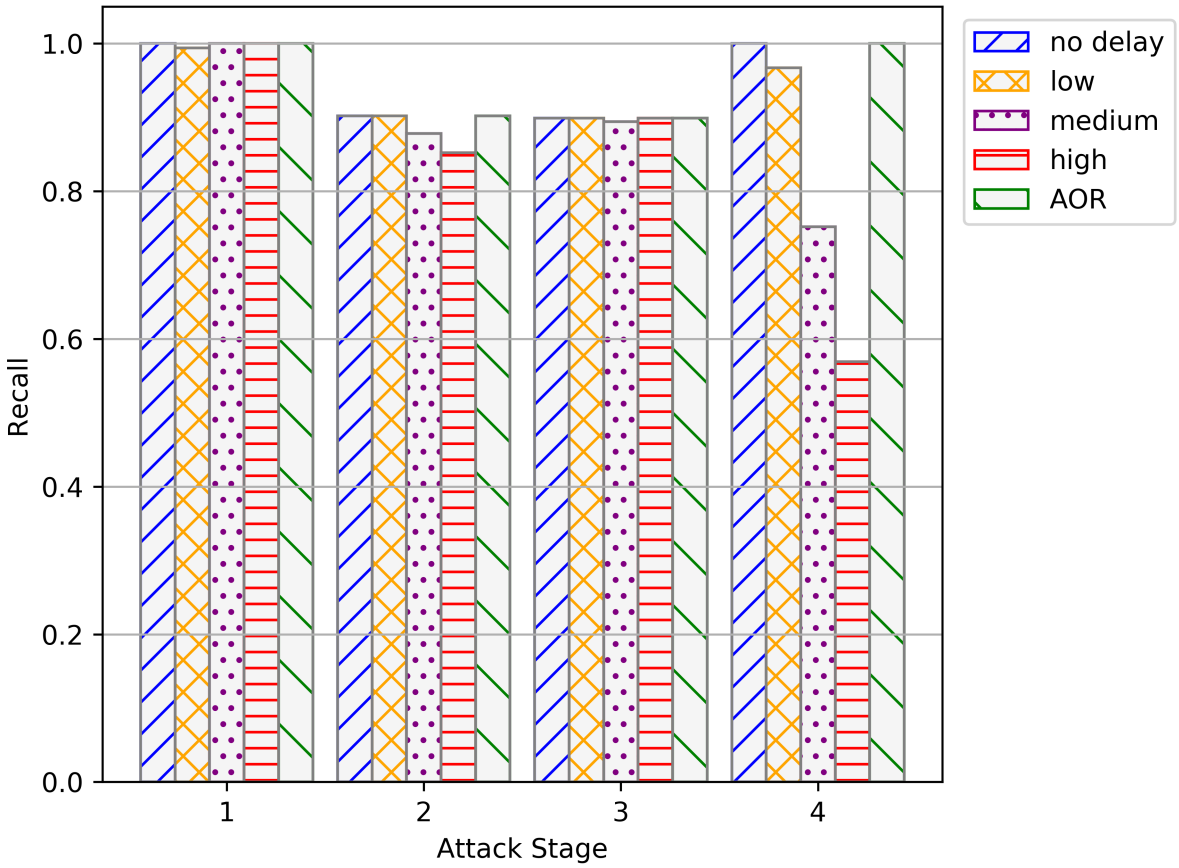


**Figure 3.28.** Prediction recall for different alert delay configurations for infiltration attack with $\mu^{10}_{max}$.

In Fig. 3.28, the prediction recall corresponding to the three delay configurations for the infiltration attack scenario with AOR operating at $\mu^{10}_{max}$ is shown. As the distributed alert reporting latency becomes more random i.e. higher value of the delay configuration, the ability of the model to predict attack stages correctly is significantly reduced, especially for those states that have fewer number of alerts, that is, attack stages with shorter length as can be observed by the prediction recall values for high delay configuration at stage 4 of the infiltration attack. The utility of AOR is evident that for all attack stages, the prediction recall becomes equal to that of the case with no delay when AOR procedure is applied on the alert streams subjected to high delay configuration.

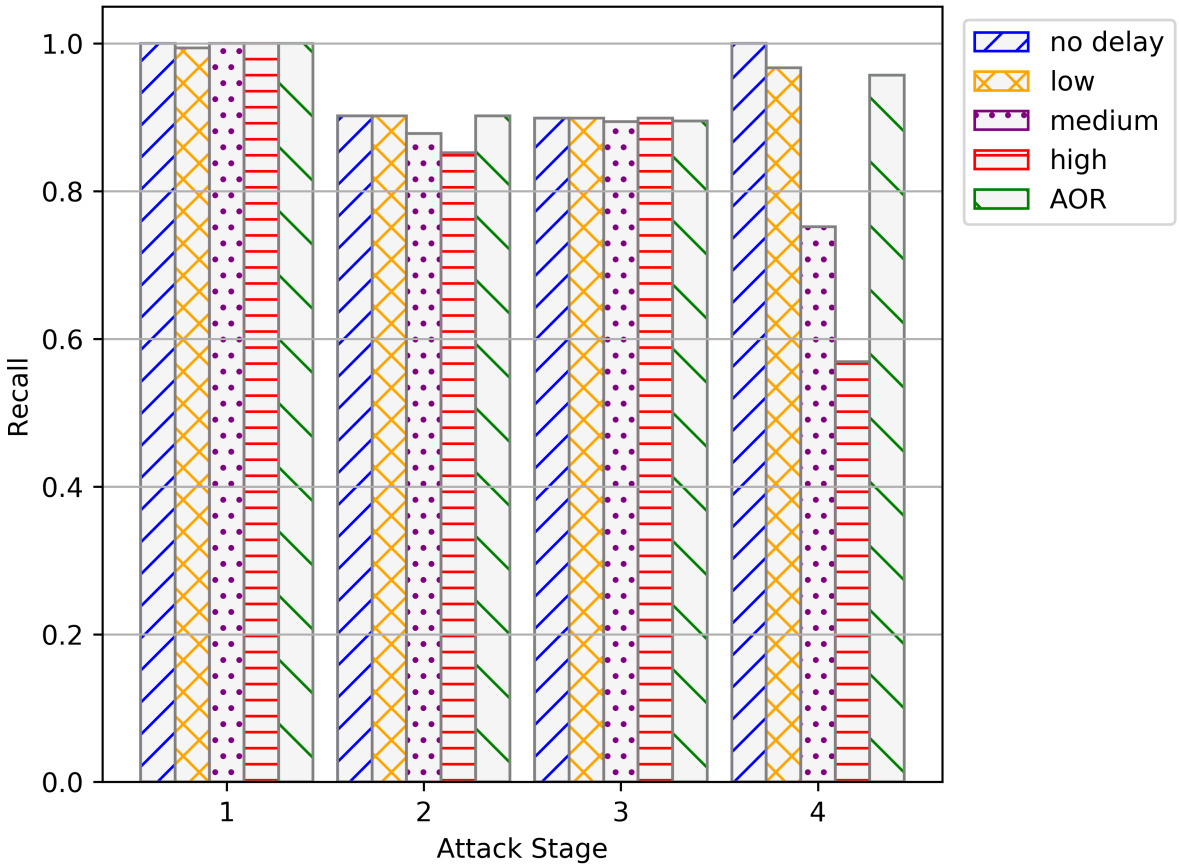


**Figure 3.29.** Prediction recall for different alert delay configurations for infiltration attack with $\mu^{30}_{max}$.

The prediction recall for the infiltration attack scenario corresponding to different alert delay configurations and AOR procedure with $\mu^{30}_{max}$ is shown in Fig. 3.29. It can be seen that as the estimate of the maximum delay experienced by alerts from different distributed sources becomes less accurate, the ability of the AOR becomes less effective in rectifying the alert order discrepancies introduced due to the distributed alert reporting mechanism. However, the prediction performance is still significantly better than the case in which the alerts are subjected to high alert delay configuration and only in stage 4 the prediction recall value is a bit less than the prediction recall at the no delay setting.

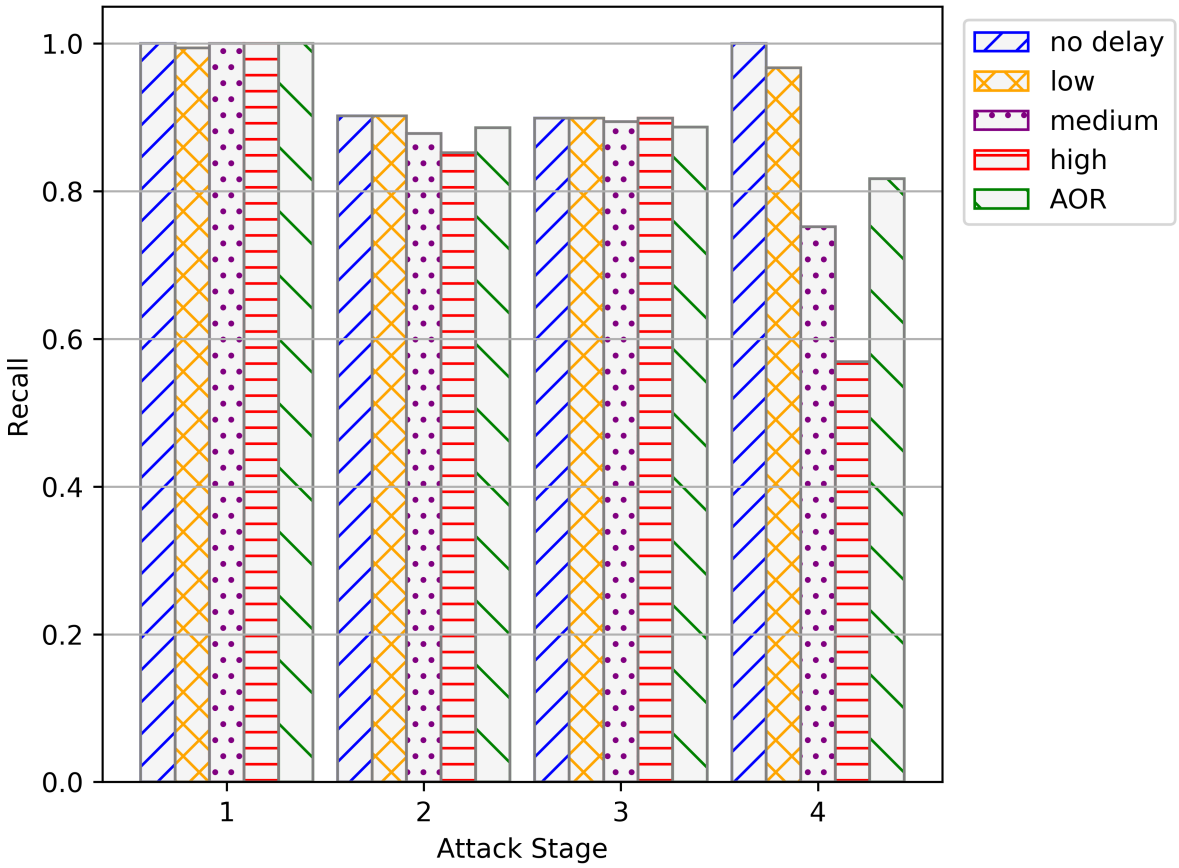


**Figure 3.30.** Prediction recall for different alert delay configurations for infiltration attack with $\mu^{50}_{max}$.

In Fig. 3.30, the prediction recall corresponding to the infiltration attack scenario for the three alert delay configurations and AOR procedure with $\mu^{50}_{max}$ is depicted. It can be seen that for attack stages 1, 2 and 3 the performance of AOR even with 50% of the maximum delay estimate, the prediction recall is similar to that in the no delay setting. In stage 4, the prediction recall with AOR procedure being applied to the alert stream is significantly better than the case with high delay configuration, but is less than the prediction recall of the no delay setting. Note that we have observed earlier that the length of attack stage 4 in the infiltration attack scenario is small as compared to other attack stages. The implication of this fact is evident that the attack stage that experiences the most reduction in prediction recall due to alert sequence distortion introduced by the distributed alert reporting mechanism is the attack stage 4.

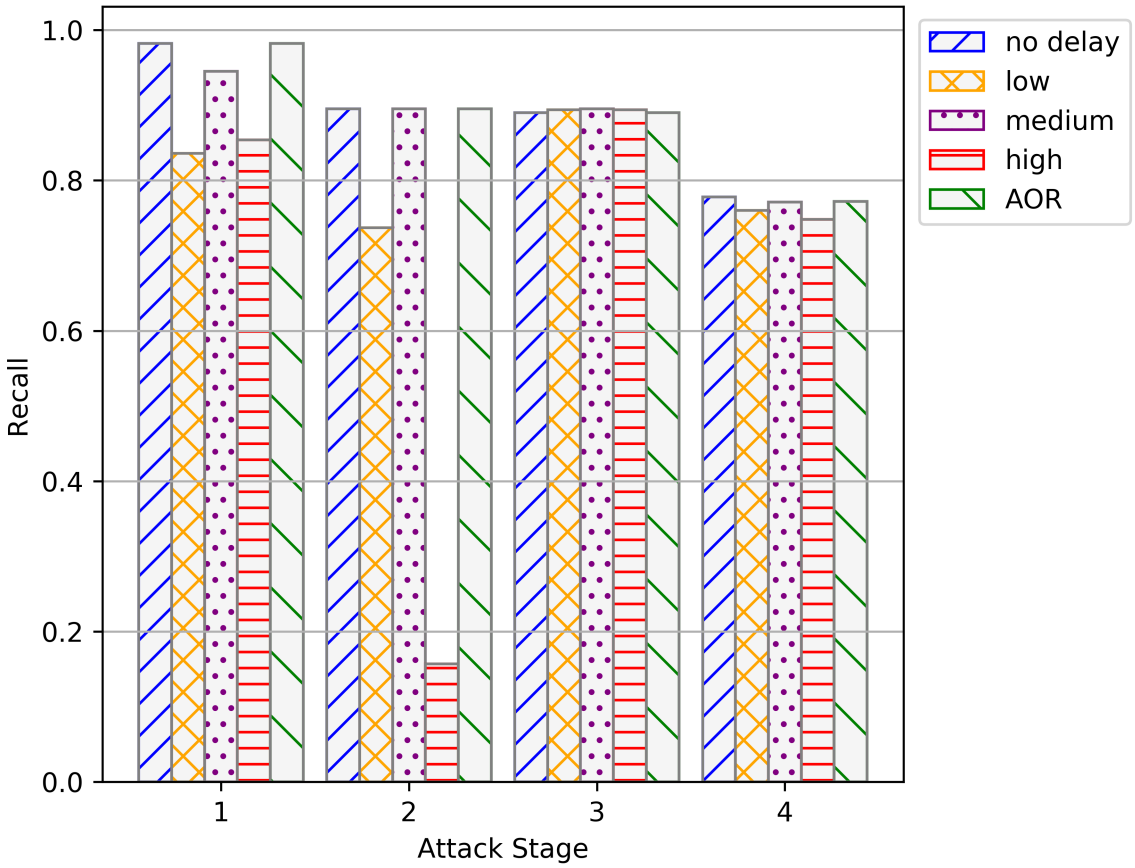


**Figure 3.31.** Prediction recall for different alert delay configurations for HTTP DoS attack with $\mu^{10}_{max}$.

In Fig. 3.31, the prediction recall value of stage 2 of the HTTP DoS attack illustrates the similar phenomenon to the infiltration attack scenario that as the distortion in the alert sequence becomes high, the ability of the prediction system to determine the attack stages correctly becomes less effective especially for attack stages that have short lengths. The effectiveness of alert stream management is also apparent as the AOR procedure with $\mu_{max}$ value 10% less than $\Delta_{max}$ restores the original order of the alert sequence without any error. It is to be noted that there is a cost associated to the application of AOR procedure as well, that every time an alert forwarding window is formed, there is a wait of $\mu_{max}$ before the alerts are sent to the Prediction Engine.

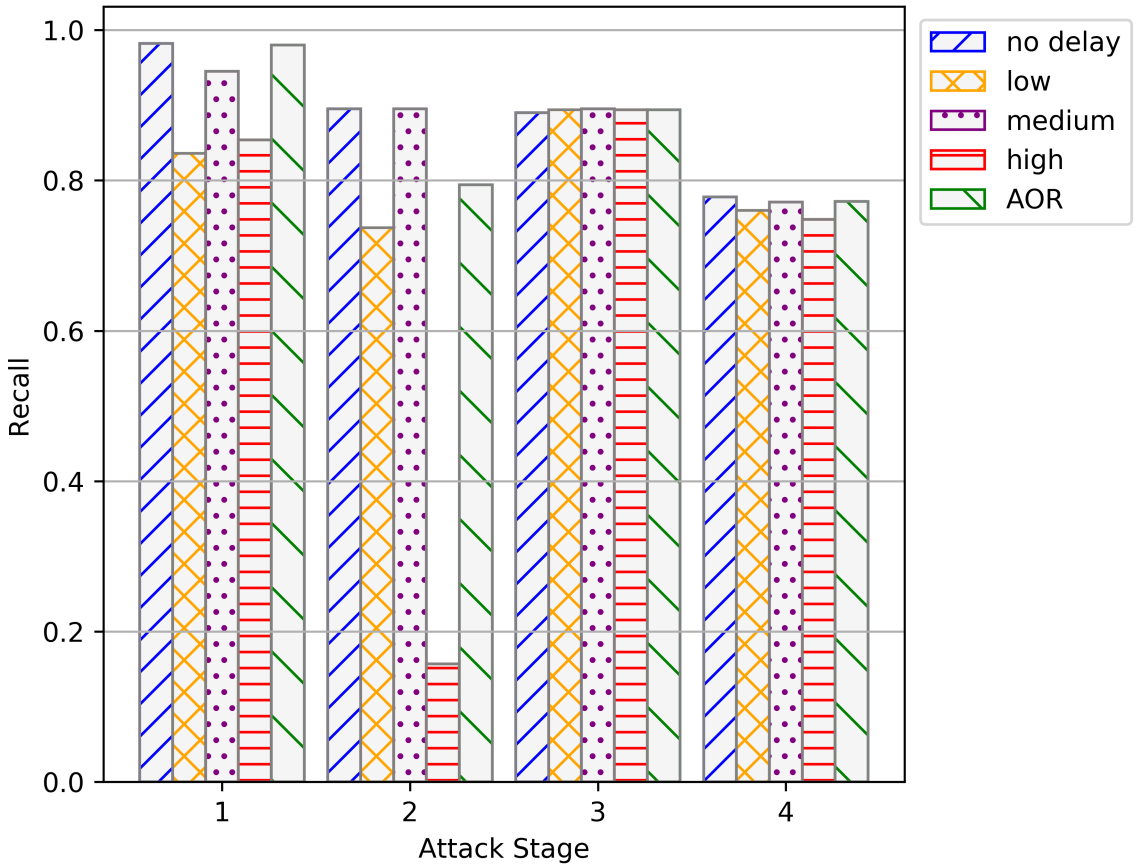


**Figure 3.32.** Prediction recall for different alert delay configurations for HTTP DoS attack with $\mu^{30}_{max}$.

The prediction recall corresponding to the HTTP DoS attack scenario for different alert delay configurations with $\mu^{30}_{max}$ delay estimate being used in the AOR procedure is shown in Fig. 3.32. It can be seen that the prediction recall corresponding to AOR and no delay case in stages 1, 3 and 4 is the same with even $\mu_{max}$ value being 30% less than $\Delta_{max}$. In stage 4, the prediction recall corresponding to AOR is significantly better than the high delay case but falls short of being equal to the no delay case. An interesting trend can be seen that the prediction recall for medium delay case is better than then low delay case in stages 1, 2 and 4. This can occur because of the possibility that the disorder introduced in the medium delay case can in fact create sequence of alerts that are considered valid alert sequences by the prediction model.

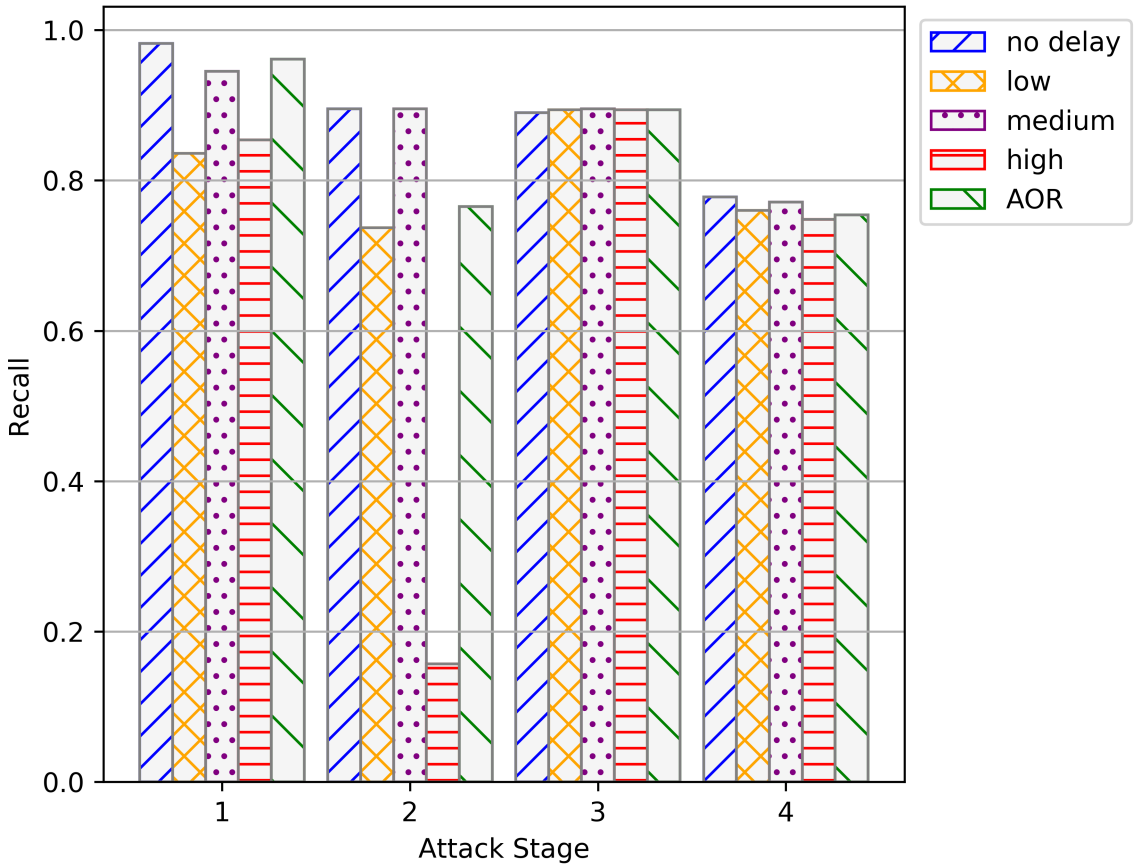


**Figure 3.33.** Prediction recall for different alert delay configurations for HTTP DoS attack with $\mu^{50}_{max}$.

In Fig. 3.33, the prediction recall for different alert delay configurations corresponding to HTTP DoS attack scenario with $\mu^{50}_{max}$ delay estimate used in the AOR procedure is shown. The prediction recall for different stages of the attack is similar to that in the case of $\mu^{50}_{max}$, but with a little bit of depreciation. In stages 1, 3 and 4, the prediction recall is slightly less in the case of AOR as compared to the no delay case. For stage 2, the prediction recall is better than that in the case of high delay configuration but is a little bit less than that in the instance of AOR with $\mu^{30}_{max}$. Note that we have mentioned earlier that the length of attack stage 2 is quite small as compared to other attack stages in the case of infiltration attack. Consequently, the impact of alert sequence distortion on the prediction recall is exorbitant for attack stage 2, especially for high delay configuration.

### Evaluation of the Rapidness in Identifying Attack Stages

Prediction recall is one aspect to quantify the prediction system's performance. In a realtime environment, it is necessary to measure how soon the prediction mechanism is able to predict attack stage transitions during the progression of the attack. The sooner the prediction apparatus is able to predict the attack stage, the more effective response can be formulated to thwart the attack. To measure the ability of the Prediction Engine to detect the correct attack stage as soon as the attack transitions from one stage to another, we propose the metric Early Detection Index (EDI) which is expressed in Eq. 3.25.

$$EDI = \frac{\sum_{\mathrm{i}=1}^{N} \frac{o_{\mathrm{i}}^{l} - o_{\mathrm{i}}^{f}}{C_{\mathrm{i}}} \varepsilon_{\mathrm{i}}}{H} \tag{3.25}$$

Where $o_{\mathrm{i}}^{l}$ represents the alert number of the last alert in attack stage i in original labeled data (ground truth) and $o_{\mathrm{i}}^{f}$ is the alert number of the first alert that the model predicts to be of attack stage i, that is, the alert that detects the beginning of attack stage i. $C_{\mathrm{i}}$ is the total number of alerts corresponding to attack stage i in the original labelled data and $\varepsilon_{\mathrm{i}}$, as mentioned earlier, is the numerical weight corresponding to the risk associated with attack stage i. Any $\varepsilon$ value can be assigned to a stage with advanced attack stages being given higher values and in our experiments the attack stage number itself is used as the $\varepsilon$ value. $H$ is the sum attack stage risk values i.e. $H = \sum_{\mathrm{i}=1}^{N} \varepsilon_{\mathrm{i}}$. The value of EDI is between 0 and 1, with a zero value or near zero value for a given prediction model illustrating that the prediction model is unable to detect any of the attack stages correctly or is only able to detect an attack stage at the last alert of that attack stage. Similarly, an EDI value of 1 shows that the prediction model is able to identify every attack stage at the first alert of the respective attack stages. We have used EDI to study the impact of alert order modification due to the alert reporting from distributed alert generation sources. The effect of the three alert delay configurations on the prediction process and the performance of alert stream management with three different estimates of the maximum alert reporting latency is measured in terms of EDI is shown in the following results.

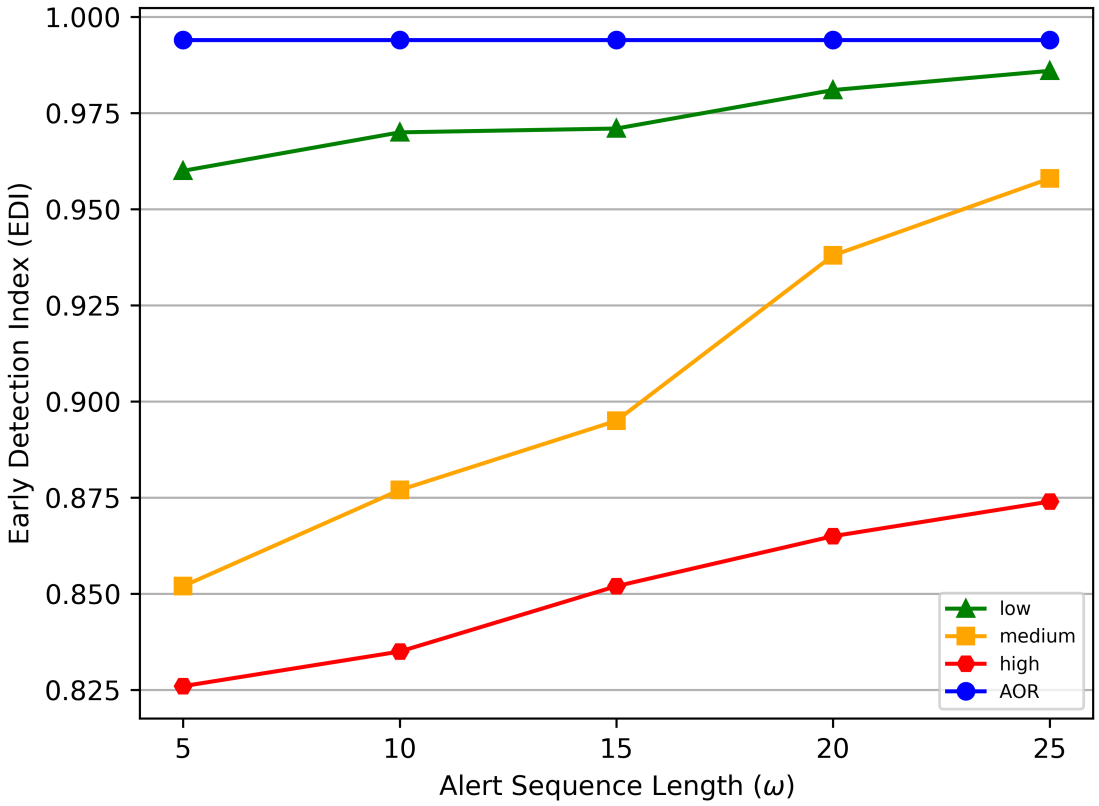


**Figure 3.34.** EDI corresponding to different alert delay configurations for infiltration attack with $\mu^{10}_{max}$.

Fig. 3.34 shows the EDI with respect to different values of $\omega$ for the three delay configurations and alert order correction using the AOR procedure in the infiltration attack scenario. It can be observed that the EDI is quite less in the case of high delay configuration as compared to medium and low delay configurations. Whereas, the EDI is near the maximum value of 1 in the case of alert stream being subjected to the AOR procedure with $\mu^{10}_{max}$. This demonstrates that the use of alert stream management not only benefits the prediction correctness but also significantly improves the ability of the Prediction Engine to detect attack stages rapidly.

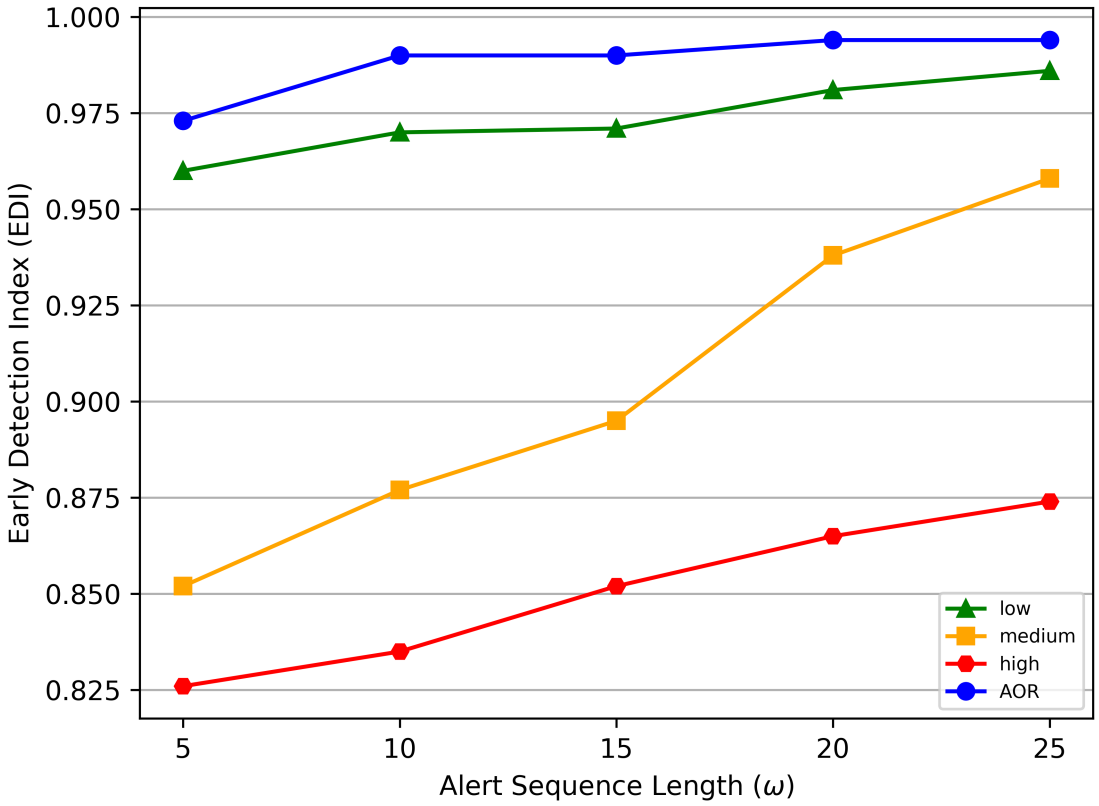


**Figure 3.35.** EDI corresponding to different alert delay configurations for infiltration attack with $\mu^{30}_{max}$.

The EDI values for different alert delay configurations and AOR with $\mu^{30}_{max}$ in the infiltration attack scenario is shown in Fig. 3.35. It can be seen that the reduction in the estimate of maximum alert reporting latency adversely effects the EDI of AOR for $\omega = 5$, however, there is improvement in EDI when the value of $\omega$ increases. The EDI in the case of $\mu^{10}_{max}$ is indifferent to the changing values of $\omega$, in contrast, it can be seen that with $\mu^{30}_{max}$ the increasing values of $\omega$ affects the EDI as well. The value of EDI in the case of $\mu^{30}_{max}$ becomes equal to that of $\mu^{10}_{max}$ for $\omega = 20$ and $\omega = 25$.

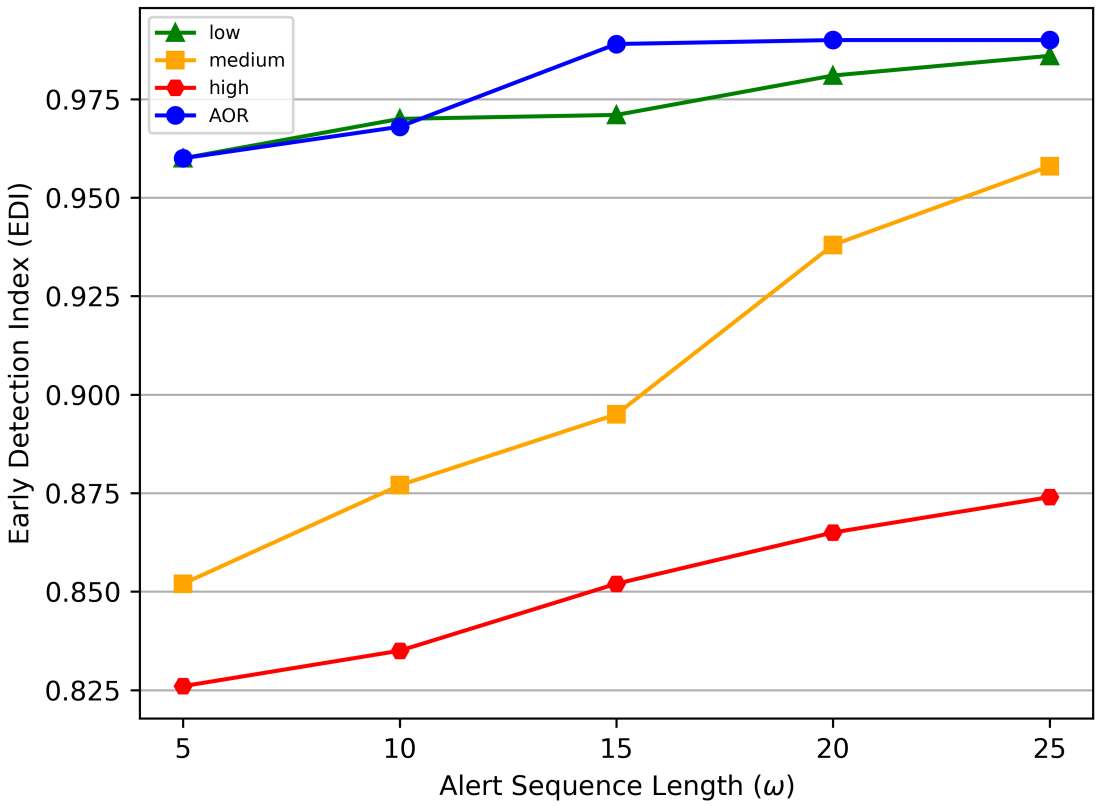


**Figure 3.36.** EDI corresponding to different alert delay configurations for infiltration attack with $\mu^{50}_{max}$.

Fig. 3.36 illustrates the EDI corresponding to the infiltration attack scenario for different alert configurations with AOR employing $\mu^{50}_{max}$, that is, the maximum alert reporting latency estimate is 50% less than the actual maximum alert reporting latency $\Delta_{max}$. It can be seen that for lower values of $\omega$, that is, for $\omega = 5$ and $\omega = 10$ EDI corresponding to AOR is equal to that of EDI for low alert delay configuration. However, with the increase in the values of $\omega$, the EDI for the alert sequence treated by the AOR procedure after being subjected to high alert delay configuration becomes significantly better. This illustrates that even with an estimate equal to just 50% of the actual alert reporting latency, the AOR significantly rectifies the alert order discrepancies introduced by the highest level of delay introduced in our experiments.

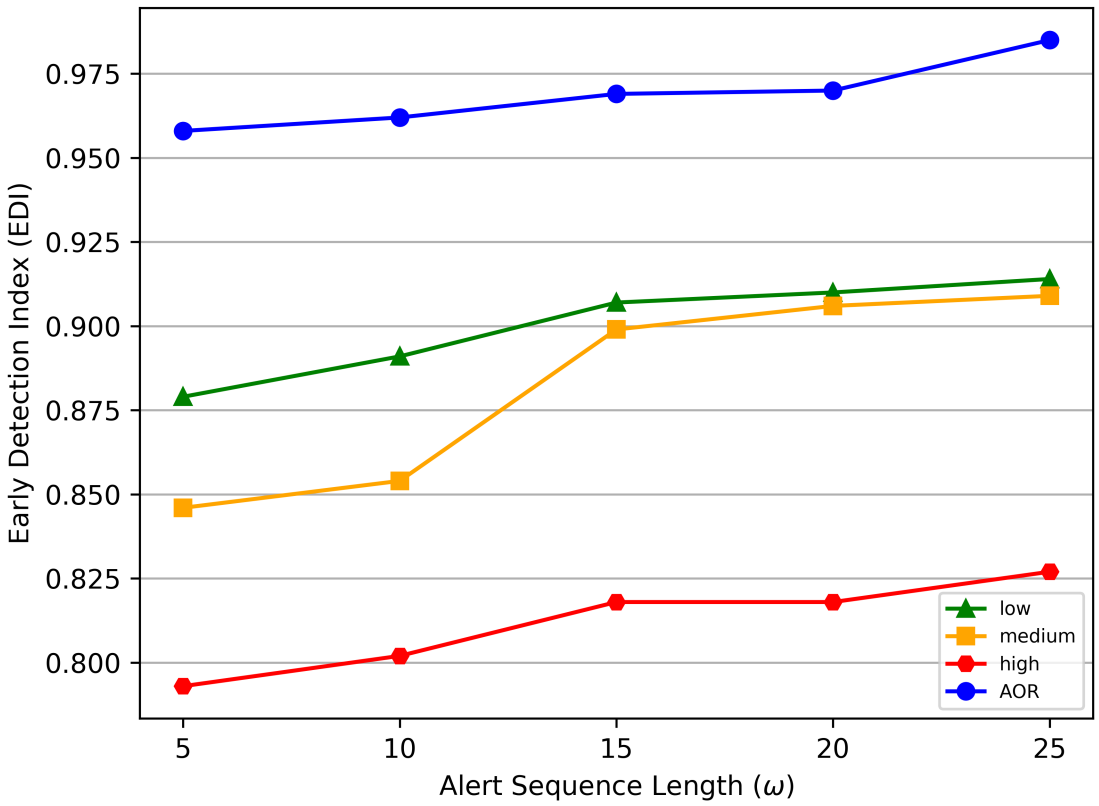


**Figure 3.37.** EDI corresponding to different alert delay configurations for HTTP DoS attack $\mu^{10}_{max}$.

The EDI values for different values of $\omega$ pertaining to HTTP DoS attack scenario for the three delay configurations of low, medium and high along with the application of AOR procedure on the alert sequence subjected to high alert delay configuration is shown in Fig. 3.37. It can be observed that the EDI values in the case of AOR and high delay configuration is similar to the infiltration attack scenario. The EDI values in the case of low delay configuration are much lower in the HTTP DoS attack scenario as compared to the infiltration attack scenario while the progression of EDI values with the increasing values of $\omega$ is slightly different in the case of HTTP DoS attack scenario.

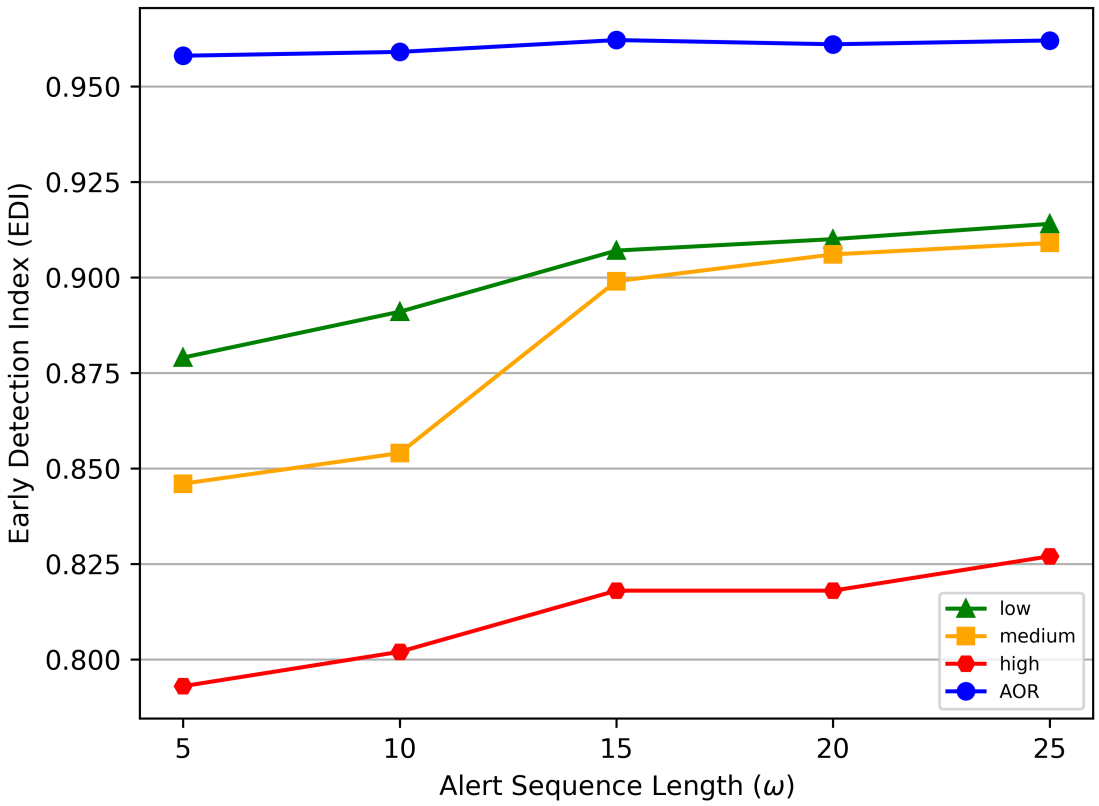


**Figure 3.38.** EDI corresponding to different alert delay configurations for HTTP DoS attack with $\mu^{30}_{max}$.

In Fig. 3.38, EDI for the HTTP DoS attack scenario corresponding to different alert delay configurations with $\mu^{30}_{max}$ being used as maximum delay estimate by AOR procedure to rectify the disorder in alert sequence due to high alert delay configuration is shown. It can be seen that the EDI for the AOR case is much better than that in the case of high delay setting. The EDI for AOR with $\mu^{30}_{max}$ is slightly less in comparison to $\mu^{10}_{max}$ and the EDI values are almost consistent with respect to the increasing values of $\omega$. It can also be observed that for alert delay configurations, the EDI value is low for $\omega = 5$ and $\omega = 10$ that increases significantly when the value of $\omega$ increases to 15, 20 and 25.

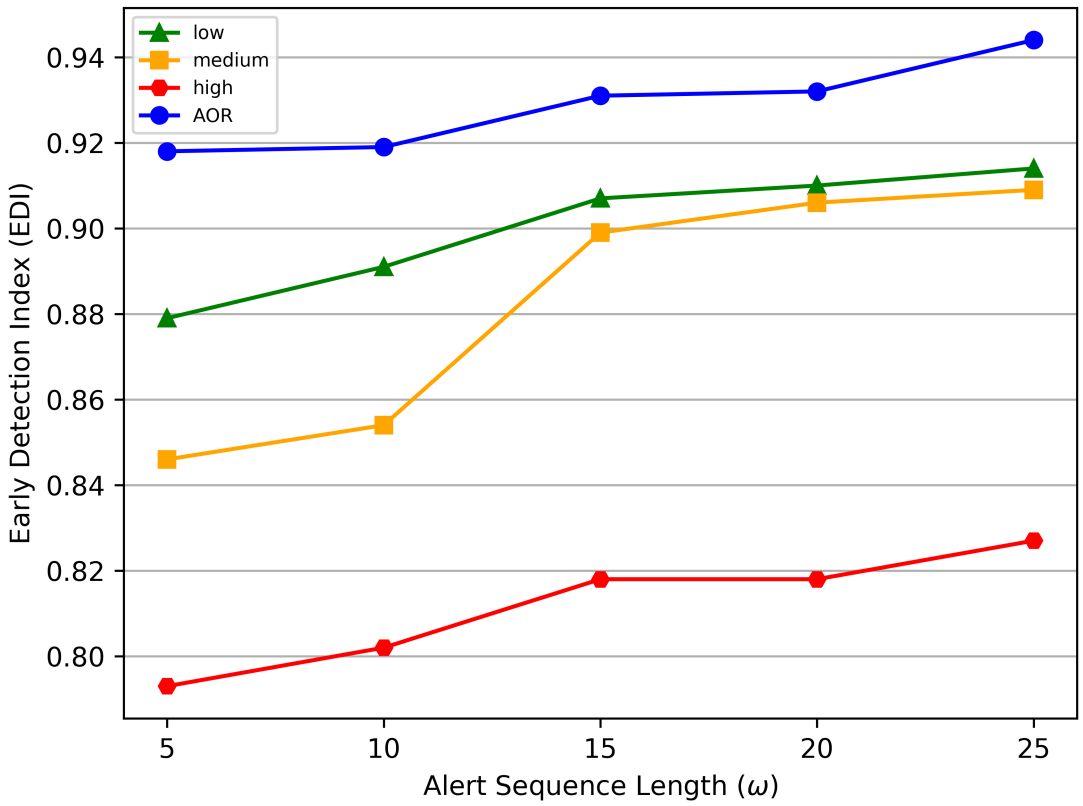


**Figure 3.39.** EDI corresponding to different alert delay configurations for HTTP DoS attack with $\mu^{50}_{max}$.

The EDI for different alert delay configurations corresponding to the HTTP DoS attack scenario with AOR procedure using $\mu^{50}_{max}$ maximum alert delay estimate is shown in Fig. 3.39. It can be seen that even with alert delay estimate being 50% less than the actual maximum alert reporting latency, the AOR procedure significantly improves the EDI as compared to the high alert delay setting. Also, the EDI values increase proportional to the values of $\omega$. In comparison the other alert delay estimates, as expected, the EDI values for $\mu^{50}_{max}$ is les than the other two. However, the EDI values for the alert sequences being subjected to the AOR procedure are significantly better than the EDI values for all delay configurations. This shows effectiveness of the AOR procedure of the ASM in its ability to identify attack stages as early as possible.

### 3.4.3 Security Analysis for Cyber Situational Awareness

We have evaluated the functionality of the Security Analyzer on the alerts generated by the infiltration attack of the ISCX-2012 dataset. In the first experiment, the network traffic is not sampled and the value of $\omega$ is fixed to 10. Also, the alerts from different surveillance zones are not subjected to any additional delay. The metrics are computed and visualized by the Security Analyzer as each of the alert is received in realtime, but here we present the plots of the complete attack.

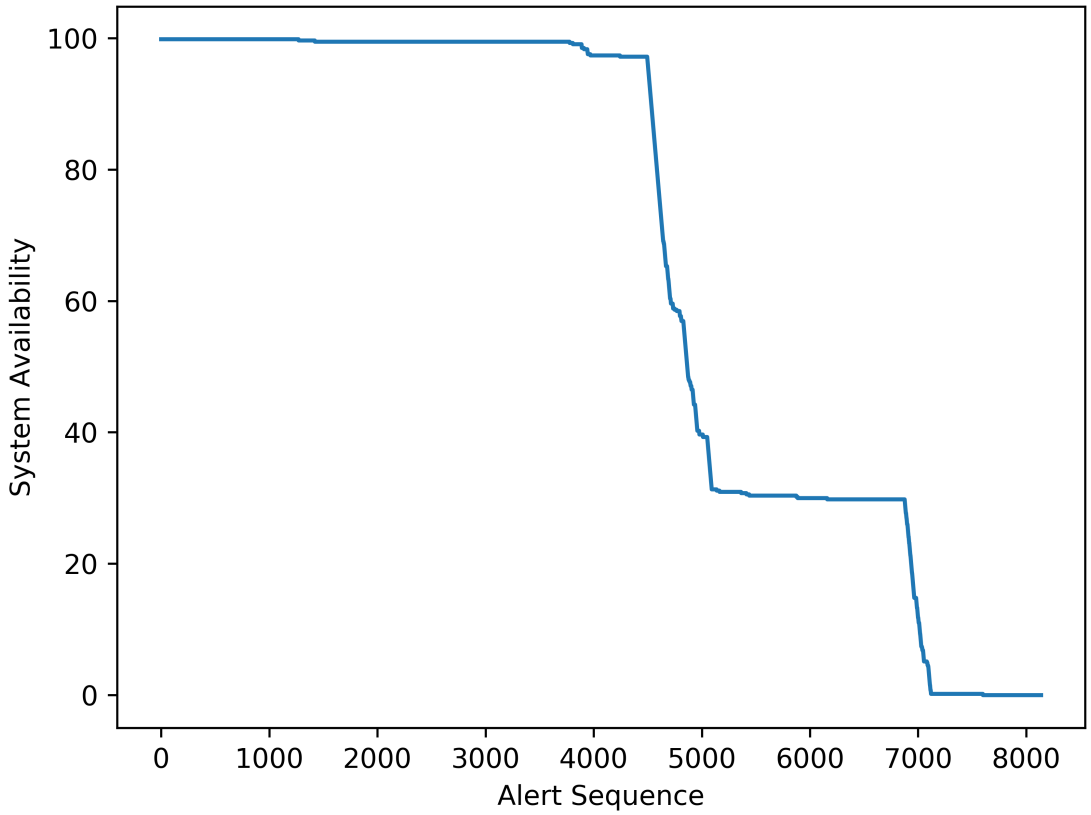


**Figure 3.40.** System availability during infiltration attack scenario.

In Fig 3.40, system availability of the network is shown with respect to the alerts. It can be seen that there is little effect on the system availability during the first half of the attack but during the second half the system availability reduces drastically.

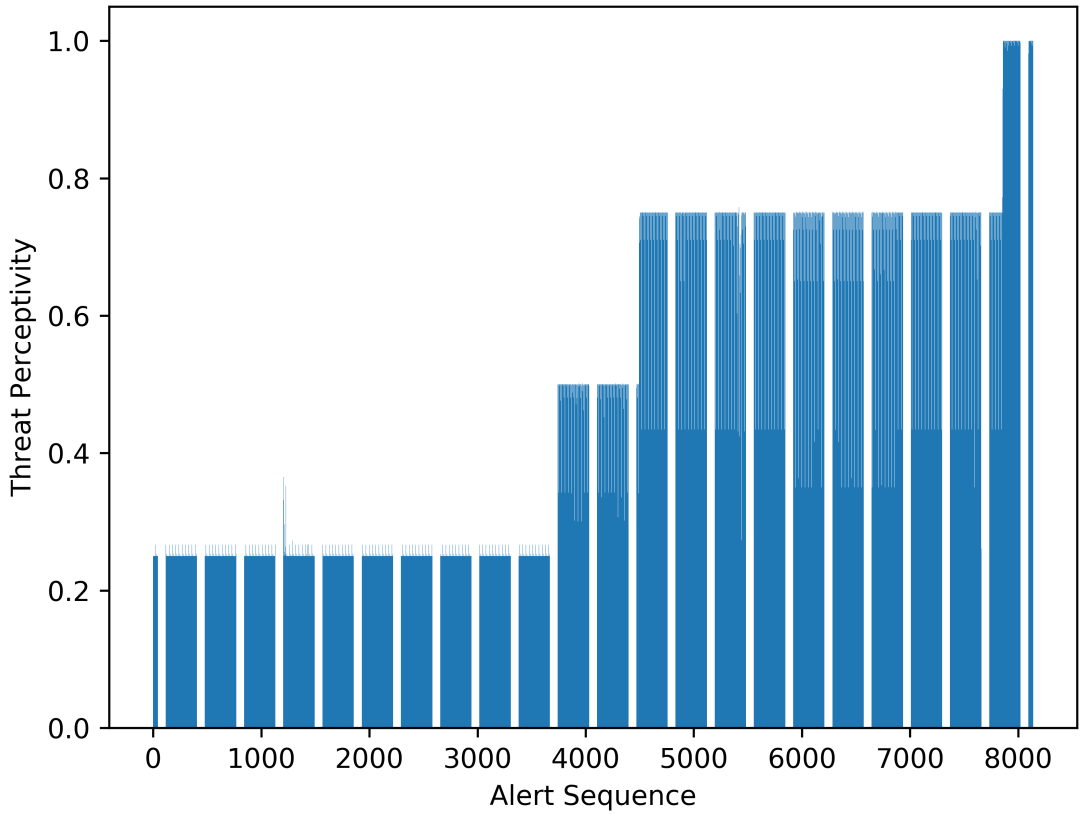


**Figure 3.41.** Threat perceptivity during infiltration attack scenario.

Threat perceptivity corresponding to the stream of incoming alerts in the infiltration attack scenario is illustrated in Fig. 3.41. As expected, the threat perceptivity increases as the attack progresses into advanced stages with mild threat perceptivity for almost the first half of the attack which increases to the maximum value of 1 when the attack is near completion. It can be seen be seen that how threat perceptivity value starts changing first after 3500 alerts and then after 4500 alerts. Threat perceptivity value effectively shows how close the attacker is at a certain point in achieving his final objectives.

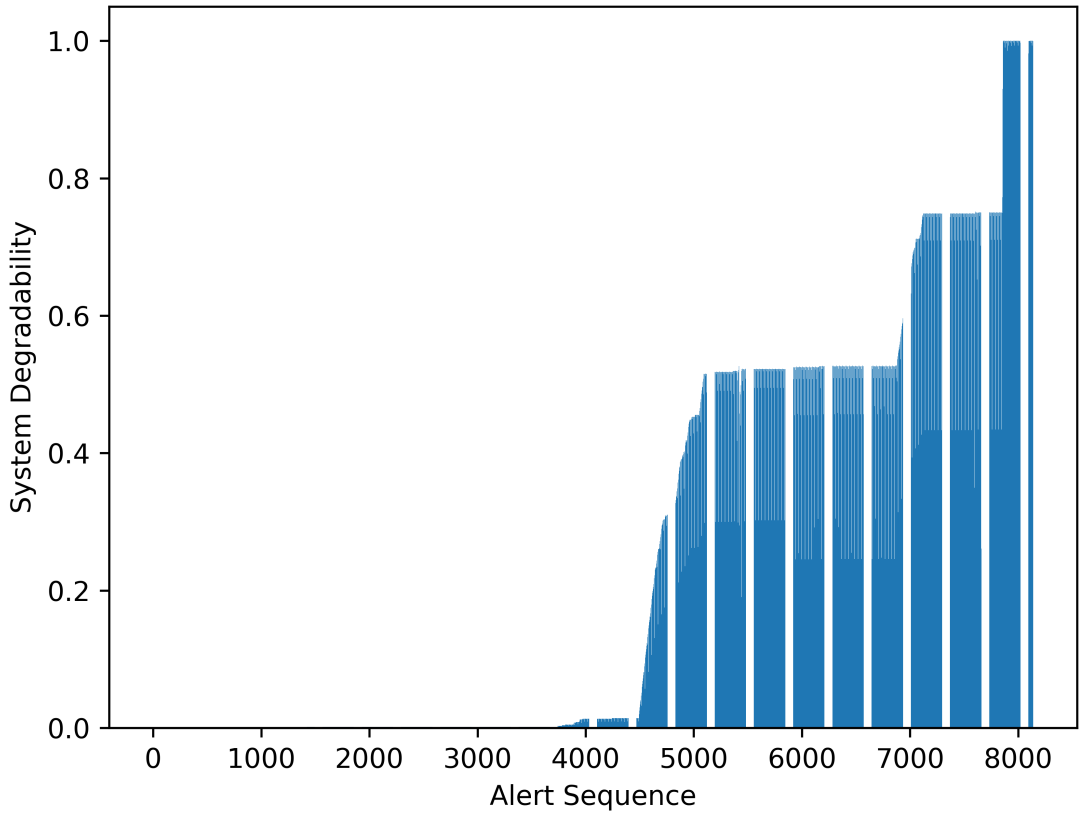


**Figure 3.42.** System degradability during infiltration attack scenario.

In Fig. 3.42, system degradability for the case of infiltration attack is presented. It can be observed that as the attack progresses, with the decreasing system availability, the system degradability increases. With the system availability almost 100% during the first half of the attack and threat perceptivity at the lowest level, the minimal values of system degradability correctly showcases that the potential impact of the system risk on the system operability is minimal. In the next half of the attack, as the system availability decreases and the threat perceptivity increases, the system degradability increases until reaching the maximum value by the end of the attack.

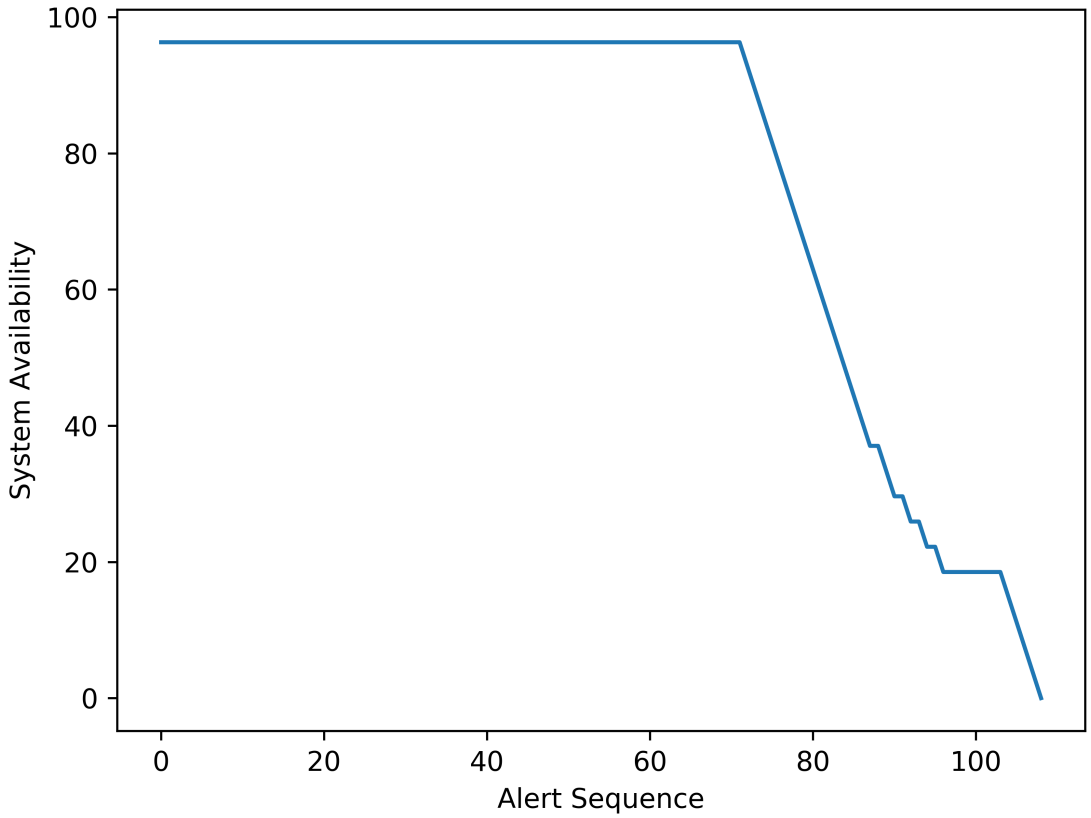


**Figure 3.43.** System availability corresponding to 1% of sampled traffic.

Fig. 3.43 presents the output of Security Analyzer with just using 1% of the network traffic that is sampled using the threat-aware sampling. In addition to exhibiting the effect of a very high sampling rate, this experimental setting reflects the impact of an evasive attack. The attack is evasive in a sense that the attacker completes his objectives by only being detected for only few of his activities. It can be observed that the system availability changes drastically to the worst levels within the last 30 reported alerts. Such a swift change in the system state shows the challenges in responding to such type of attacks as it is significantly difficult for any threat response system to counter an attack that produces very few traces.

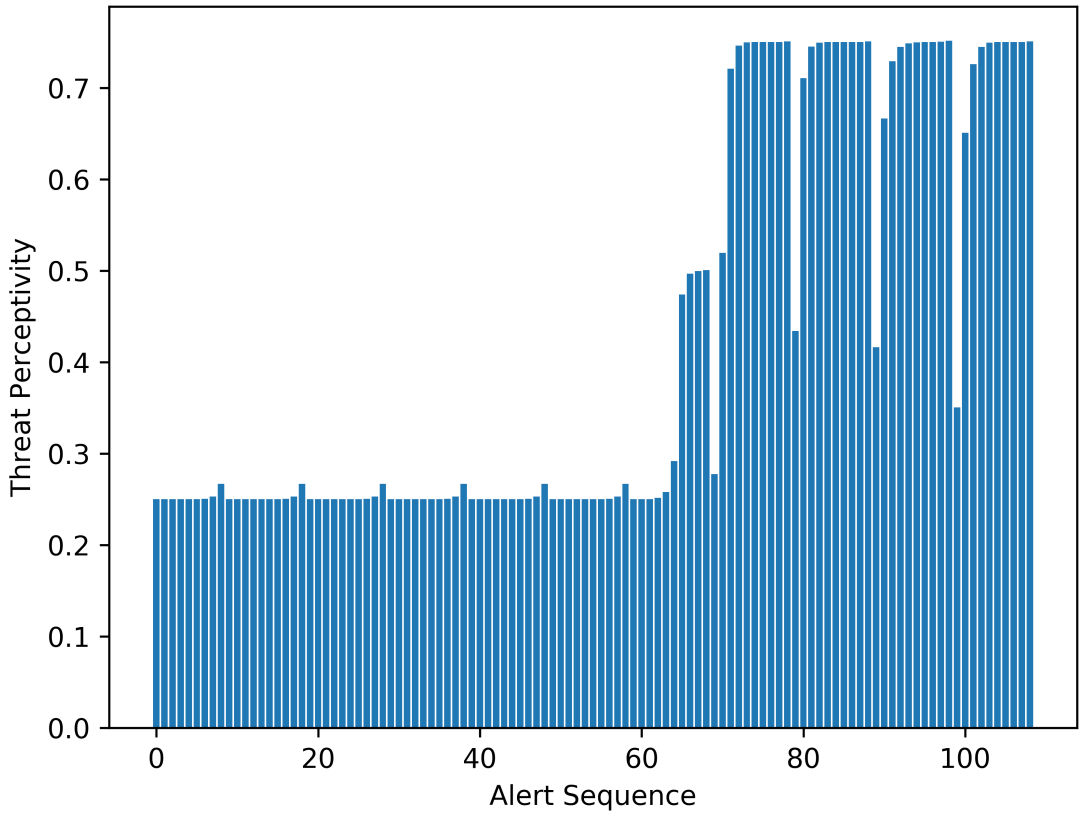


**Figure 3.44.** Threat perceptivity corresponding to 1% of sampled traffic.

Similar to the impact of a very high sampling rate that also mimics the behavior of evasive attacks on system availability, threat perceptivity plot shows an irregular pattern within the last 30 alerts of the infiltration attack data with 1% of sampled traffic as shown in Fig. 3.44. It can be seen that during the last 30 alerts, threat perceptivity increases sharply, decreases at some points and then increases again. This behavior effectively shows the limitations of threat detection and prediction systems designed to assist a manual response. As a human operator will be unable to understand the situation with drastically changing dynamics in a short period of time resulting in the successful completion of such attack types.

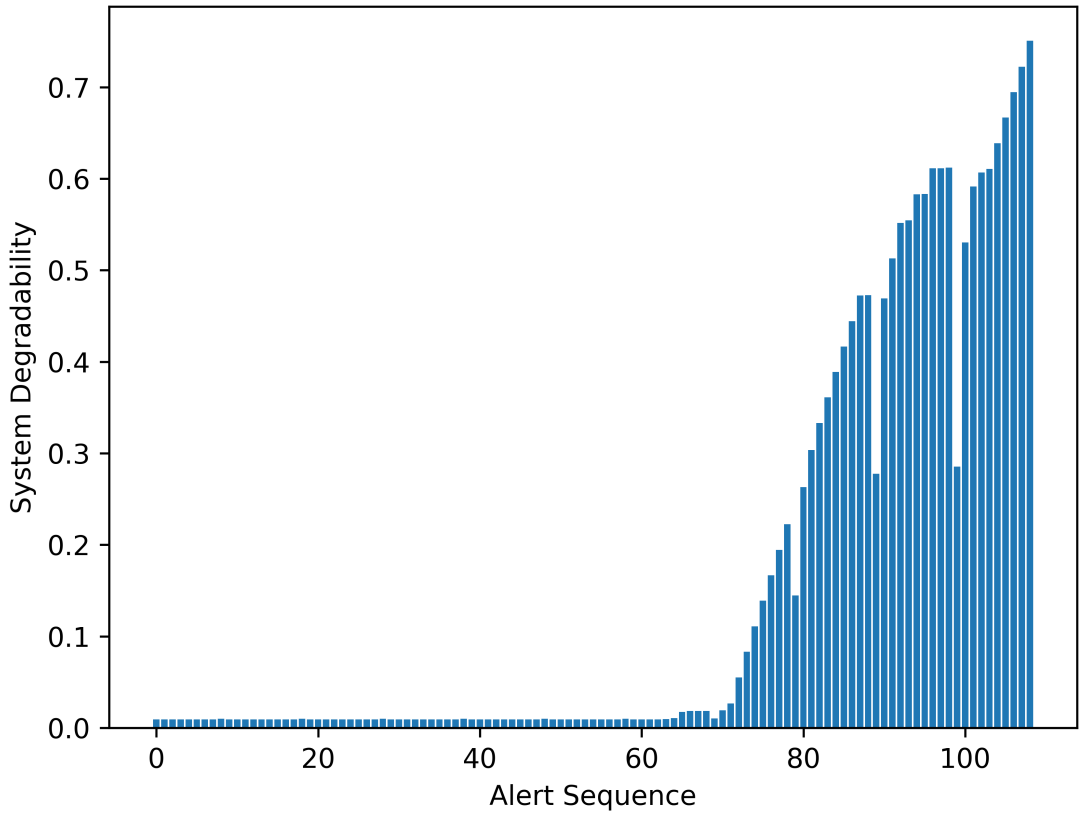


**Figure 3.45.** System degradability corresponding to 1% of sampled traffic.

The progression of system degradability is not different from system availability and threat perceptivity for the 1% sampled data of the infiltration attack scenario. During the last 30 alerts it can be seen that system degradability increases quickly, decrease at some points and then increases drastically. This shows the limitation of Security Analyzer in assisting human-driven response operations and reaffirms the proposition that automated response mechanisms are better suited to foil such attack types. In Chapter 4 of the dissertation, we propose an automated threat response and recovery architecture that is designed to counter fast propagating evasive attacks in a timely manner.

# 4. TRAP: A PARTITION-DRIVEN INTEGRATED SECURITY ARCHITECTURE FOR CYBER-BASED SYSTEMS

The structure of TRAP and its interaction with a general CBS framework is depicted in Figure 4.1. TRAP is general enough to be used for Cyber-Physical System (CPS) applications as well. The efficacy of TRAP is shown for Advanced Metering Infrastructure security which illustrates that TRAP can operate with any CBS and can be extended to even more complex systems like CPS. TRAP is by design general and system agnostic. In this chapter the terms CBS and CPS are used interchangeably since both are distributed systems of heterogeneous components or hosts and the objective of the architecture is to contain the damage by an automated response and recover the system in an automated fashion at the same time. TRAP has four constituent systems: partition manager (PAM), intrusion response system (IRS), intrusion recovery system (IRC) and performance monitor (PEM). TRAP utilizes the service of an intrusion detection system (IDS) that generates alerts upon detection of a cyber-attack. The architecture also uses the operational logs of CPS and assumes that they are updated in realtime with CBS component activities such as data flow between components with timestamp. Furthermore, it is assumed that the CBS logging system is part of the CBS's trusted computing base (TCB), as correct functioning of the logging system is essential for the architecture's recovery operations. The implementation of the core modules of the security architecture is non-trivial but it can be standardized. The effort in deployment of the security architecture is largely dependent on the available flexibility in interfacing with the CBS control system. We discuss the constituent systems of the architecture and their internal modules in detail starting with a discussion on the intrusion detection process.

## 4.1 Intrusion Detection System

An extensive discussion about the functioning of the IDS is provided in the previous chapter. TRAP starts its operations the moment IDS makes detection and identifies the components of the CBS that got compromised by an intrusion.

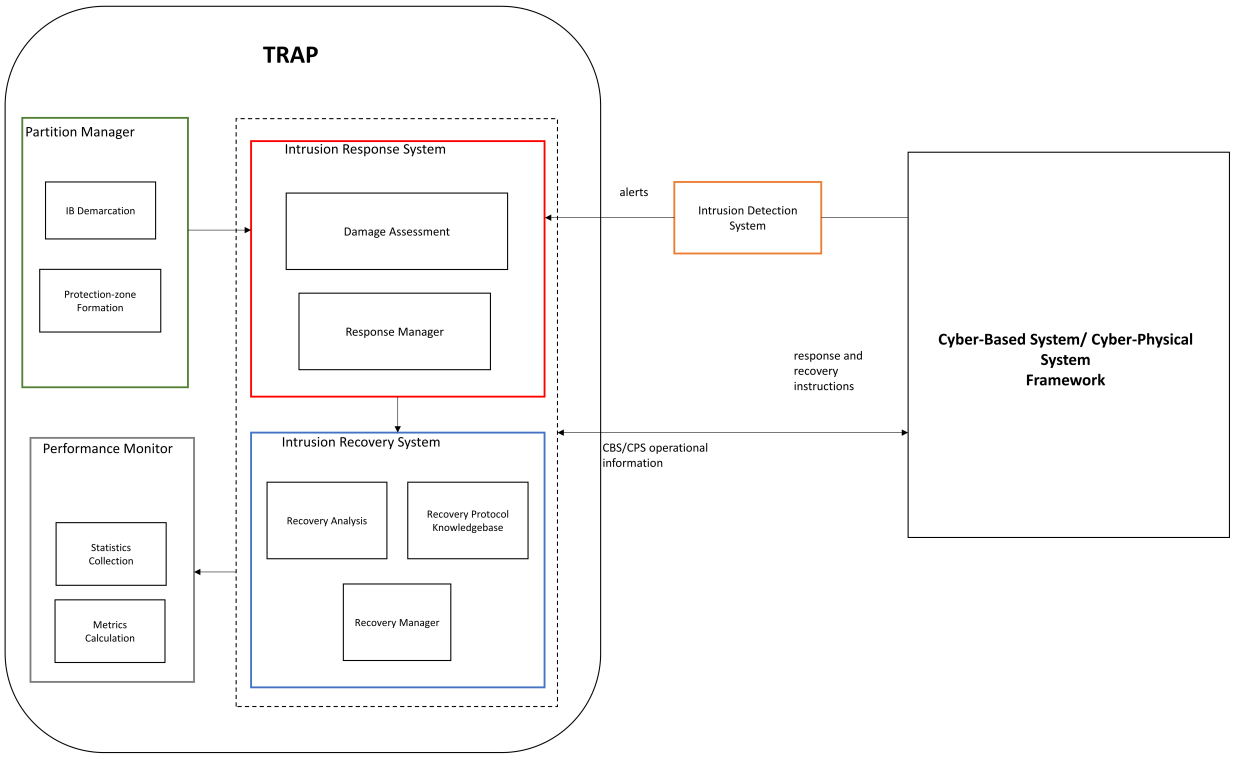


**Figure 4.1.** TRAP with its constituent systems and interaction with a CBS.

## 4.2 Partition Manager

PAM partitions the CBS components in two levels. The first level of partitioning involves demarcation of intrusion boundaries (IBs). The objective of the IBs is to contain damage propagation across IBs. The second level of partitioning involves the formation of protection-zones. The objective of protection-zones is that within an IB, components affected by the attack needs to be isolated quickly so that the impact of the attack is confined to affected components only.

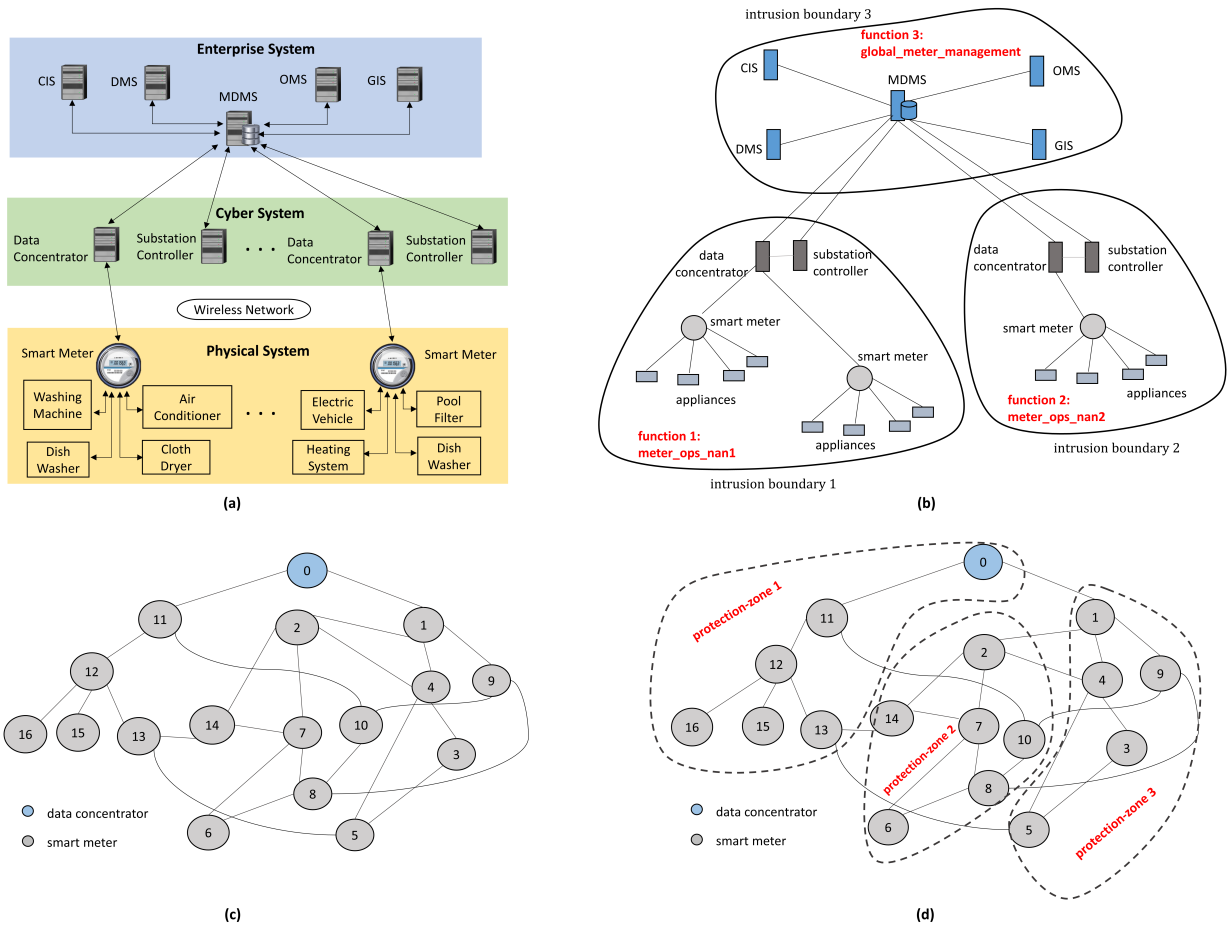


**Figure 4.2.** Partitioning process for AMI. (a) Multi-layered system of systems AMI architecture. (b) Demarcation of IBs on a given AMI topology. (c) AMI topological structure in an IB. (d) Partitioning of the AMI topological structure to form protection-zones.

### 4.2.1 Intrusion Boundary Demarcation

The CBS components that collaborate to provide a specific CBS function are included in the same IB. It is possible that a CBS component handles multiple functions while the same CBS component is shared among multiples IBs. An IB can encompass more than one functions as well. To create IBs, CBS topological structure (component connectivity and reachability) and CBS component to functionality mapping is used, and are assumed to be available to the architecture. Figure 4.2(a) shows advanced metering infrastructure (AMI) architecture and Figure 4.2(b) shows the division of AMI components in different IBs. The

smart meters that are connected to the same data concentrator perform meter_ops_nan function, and form a neighborhood area network (NAN). Therefore, they are part of the same IB. Similarly, the AMI components at the enterprise system layer such as meter data management system (MDMS), consumer information system (CIS), outage management system (OMS), distribution management system (DMS) and geographic information system (GIS) are mapped to global_meter_management function, and can be placed in the same IB. The idea behind IBs is that an attack inside an IB is fully contained in it by design. The communication among different IBs is carried out by dedicated links or in some cases through shared components. The inter-IB communication can be ensured to be not malicious by applying a positive firewall filter policy on the inter-IB communication links or on the outgoing traffic from the shared component. To minimize the firewall deployment overhead, the number of inter-IB communication links should be minimum. To find an optimal IB configuration, an optimization problem similar to classical graph partitioning problem [86] can be formulated that minimizes the edge cut among IBs.

### 4.2.2 Protection-zone Formation

IBs can contain thousands of CPS components and an attack starting from a single component in an IB can potentially compromise all of the components in that IB due to functional dependencies. Figure 4.2(c) shows an example AMI topology enclosed in an IB. In order to contain the damage within an IB, we introduce a second level of partitions known as protection-zones. The PAM groups CPS components of an IB in almost fixed-size protection-zones such that the chances of inter protection-zone damage propagation are minimized. The response system of the architecture utilizes the protection-zone information to effectively localize the damage in an IB. The protection-zone creation process is formulated as a single-objective multiple constraints optimization problem with the goal to minimize the inter protection-zone communication.

The CPS components in an IB and the interconnections among them can be modeled as an unweighted non-directional graph $P$ as shown in Figure 4.2(c). Let $P = V, E$ where $V$ is the set of vertices modeling CPS components and $E$ is the set of edges modeling the

communication links between CPS components. Let $P_{\mathrm{i}}$ be the $ith$ subgraph, such that $P = \bigcup_{\mathrm{i}=1}^{k}$ and $P_{\mathrm{i}} \bigcap P_{\mathrm{j}} = \phi$. For each $v \epsilon P_{\mathrm{i}}$, let $Q_{\mathrm{i}} = \{P_{\mathrm{j}} \mid \exists\, \mathrm{e}(v,u)\, and\ u\, \epsilon\, P_{\mathrm{j}}\ where\ \mathrm{j} \neq 1\}$ i.e. $Q_{\mathrm{i}}$ is the set of neighboring subgraphs that contain a vertex $u$ that is adjacent to $v \epsilon P_{\mathrm{i}}$ whereas there exists an edge $\mathrm{e}(v,u)$ between $u$ and $v$.

The problem is to partition the graph $P$ into $k$ balanced subgraphs with minimum edge cut such that a vertex at the boundary of a subgraph is at most connected to $1+\eta$ other subgraphs. Let $\bigotimes_{\mathrm{ij}} = \{\, \mathrm{e}(v_{\mathrm{i}}, v_{\mathrm{j}}) \mid v_{\mathrm{i}}\, \epsilon\, P_l\ and\ v_{\mathrm{j}}\, \epsilon\, P_k, \forall\, k \neq l\}$ i.e. $\bigotimes_{\mathrm{ij}}$ is the set of edges extending from one subgraph to another. The optimization problem is as follows.

$$min \sum_{\mathrm{i}\neq\mathrm{j}} \mid \bigotimes_{\mathrm{ij}} \mid \tag{4.1a}$$

$$|P_{\mathrm{i}}| \leq (1+\epsilon)\, \frac{|V|}{k},\ \forall_{\mathrm{i}} \tag{4.1b}$$

$$|Q_{\mathrm{i}}| \leq (1+\eta)\, \forall_{\mathrm{i}} \tag{4.1c}$$

The objective function minimized the number of communication links between two different subgraphs or protection-zones. The damage propagates through the communication links between CPS components, therefore minimizing the communication across protection-zones reduces the chances of inter-protection-zone damage propagation. Moreover, the Constraint 4.1b balances the sizes of the partitions as small-sized partitions are unable to restrict the damage spread in one protection-zone and large sized partitions allow damage to propagate to a large number CPS components without any hindrance. Constraint 4.1c ensures that a CPS component in a protection-zone is at most connected to components in $1+\eta$ protection-zones. The intuition is that, if a component that is connected to components in many different protection-zones is attacked then damage could quickly propagate to other protection-zones. The parameters $\epsilon$ and $\eta$ are used to relax the constraints, the lower values of these parameters the better expected performance in terms damage containment, but usually if the constraints are too tight then there might not be a feasible solution for a given CPS topological structure. This graph decomposition problem is similar to the classical

uniform graph partitioning problem [86] and the PAM employs a greedy algorithm to create protection-zones (partitions) of similar sizes.

PAM maintains information about the respective member components of protection-zones and the boundary components of all protection-zones. A CPS component that has connections to components in other protection-zones is a boundary component. Figure 4.2(d) shows the creation of protection-zones within an IB for AMI as an example CPS.

## 4.3 Intrusion Response System

IRS is activated by the IDS upon detection of an intrusion. IRS generates response for the cyber-attack by utilizing the information provided by the IDS and PAM. The functionality of IRS is described as follows.

### 4.3.1 Damage Assessment

The damage assessment (DA) module uses protection-zone information provided by PAM to identify the protection-zones where damage has been spread. PAM shares two data structures with the DA module. The first one maps every CBS component to a protection-zone and is named pz_component_list. The second data structure contains information about the boundary components of the protection-zones and is named pz_boundary_set. The IDS detects intrusions in the CBS and generates alerts that specifies the CBS components that are compromised. The pseudocode in Algorithm 5 describes the damage assessment process.

**Algorithm 5** Damage Assessment

**Input:** *pz_componnent_list*, *pz_boundary_set*, *ids_alert*

**Output:** *response_set*

1: **for** each ids_alert of a corrupted component **do**
2: find protection-zone P of componenet using pz_componenent_list;
3: identify boundary components of P using pz_boundary_set;
4: include the identified boundary components in response_set;
5: **end for**

For each alert received by the DA module, response_set is computed. The components that need to be isolated from the rest of CBS are in the response_set. It is possible that by the time IDS detects an intrusion the damage has already spread to other protection-zones. In that case, the IDS keeps on generating alerts as more CBS components get compromised and the DA module keeps generating response_sets and provide them to the response manager module until the damage is contained.

### 4.3.2 Response Manager

Response manager (RSM) coordinates the operations of DA and interfaces with the CBS control system. RSM starts its operations the moment it receives an alert from IDS. First, RSM halts the operation of the compromised components. Subsequently, it sends instructions to CBS control system to disconnect the CBS components identified in response_set. The goal here is to stop inter protection-zone damage propagation. This action might temporarily halt certain CBS functions or services, resulting in a brief unavailability of those functions in exchange for the benefit of damage containment. RSM continues this process until there are no more alerts from the IDS, indicating the damage has been successfully contained. Subsequently, RSM activates IRC to start the recovery process. Additionally, RSM sends IRS operational information to PEM.

## 4.4 Intrusion Recovery System

IRC recovers the system in an automated fashion by determining the exact extent of damage, and performing corrective actions accordingly. The architecture responds first and recovers afterwards, because the recovery is a time consuming process while response can be deployed quickly after the detection of the cyber-attack. We explain the functions of several IRC modules briefly.

### 4.4.1 Recovery Analysis

Recovery analysis (RA) module assesses the extent of damage by precisely identifying the compromised components of CBS. For an IDS alert, the recovery process is initiated

after completion of the response process. RA module analyzes operational logs of the CBS that contains detailed information about the activities of CBS components. The activity of interest is usually the data flow information between CBS components. RA module uses the IDS alert information to find the start time of the cyber-attack (Ts) and the compromised CPS component. RA module then starts analyzing the CPS operational log to identify all of the components to which the compromised component communicated and includes them in a data structure known as repair_set. The RA process is described in Algorithm 6.

**Algorithm 6** Recovery Analysis

**Input:** *ids_alert*, *op_log*

**Output:** *repair_set*

1: **for** each ids_alert of a corrupted component **do**
2: include compromised component in repair_set;
3: **end for**
4: **for** each new entry s in repair_set **do**
5: identify components r in op_log that communicated with s after $T_s$;
6: include component c $\in$ r in repair_set, if c $\notin$ repair_set;
7: **end for**

### 4.4.2 Recovery Protocol Knowledgebase

Once the RA module identifies the compromised components, the Recovery Protocol Knowledgebase (RPK) module determines the recovery protocol to recover them. RPK module maintains a knowledgebase that provides procedures that needs to be followed to recover the compromised CBS components. For example, if the data management server is compromised then the recovery techniques mentioned in [87] can be employed. Similarly, if a field device is compromised then a redundant device could be activated, if available. Otherwise, if the device was providing essential data to a controller then instructions can be given to the controller to use simulated data till the device is restored manually.

### 4.4.3 Recovery Manager

Similar to RSM, the recovery manager (RCM) interfaces with CBS control system. RCM submits recovery instructions to take corrective actions based on the information provided by the RPK module, for a particular attack. Notice that RCM might not be able to completely recover the CBS functionality in an automated manner due limitations of the CBS infrastructure or unavailability of appropriate recovery protocol in RPK for a particular component. In that case, RCM only disables the corrupted component, and generates an alert to recover the system manually. Once the recovery process of the system is complete, the normal communication of all the CPS components is restored. RCM also records the activities of IRC and forwards this information to PEM.

## 4.5 Performance Monitor

PEM provides an overview of response and recovery system performance by collecting statistics from IRS and IRC, and calculating different performance metrics. The performance monitor has two modules that are discussed briefly.

### 4.5.1 Statistics Collection

Information from different systems of the proposed security architecture is collected and analyzed by statistics collection module. The received information can be in the form of extensive log files, therefore this module extracts useful data that is used by metrics calculation module.

### 4.5.2 Metrics Calculation

We introduce operational availability as a measure of serviceable CBS functions, and derive damage extent from operational availability to reflect the impairment of CBS functions caused by an attack. To monitor the impact of response operations on the communication of CBS components, the metric communication impediment is proposed. The metrics are defined as follows.

- **Operational availability**: the available CBS functionality in terms of percentage of components unaffected in an attack for a CBS function.
- **Damage extent**: the compromised CBS functionality in terms of percentage of components damaged in an attack for a CBS function.
- **Communication impediment**: the observed limitation in the communication of CBS components due to response operations in terms of the percentage of components experiencing restricted communication.

Note that the metrics operational availability and damage extent concern a particular CBS function. The overall CBS availability and damage extent can be found out by averaging these metrics for all CBS functions.

## 4.6 Simulations and Evaluation

We have used AMI as the target CPS application to evaluate the performance of TRAP. Note that as mentioned before to show the generality of TRAP we have extended its application in CPS which similar to CBS but with added complexity. We first discuss the threat model concerning AMI followed by a discussion on the simulation tool and implementation. Finally, we present our experimental setup and compare the performance of TRAP with the commonly adapted intrusion prevention approach.

### 4.6.1 Threat Model and Implementation

We have employed the electricity pricing manipulation cyber-attacks on AMI as the attack scenario in our experiments. The attacker tries to exploit access vulnerabilities of smart meters through a direct physical link or through the AMI. The objective of the attacker in electricity pricing manipulation cyber-attack is to disrupt the distribution system of the power grid such that electricity supply to the targeted regions is suspended. The attacker maliciously gains access to a smart meter and usually injects a malware that could be propagated through the network. Such cyber-attacks can be detected by modeling the state of each meter in a distributed fashion with a centralized decision making mechanism [88]. In

AMI, smart meters are connected to the data concentrator through a mesh network that forms a neighborhood area network (NAN). The target of the malware is the guideline price information that is used by smart controller to manage the usage of different appliances to reduce the electricity bill. The smart controller when observes a lower price at a certain time of the day, schedules the usage of many appliances, resulting in the increase in energy consumption of that particular consumer. The consumers in a NAN are usually connected to the same bus used by distribution system of the power grid. If the attacker succeeds to manipulate the pricing information in a large number of smart meters one after the other, the transmission line that delivers power to the bus can trip. This can cause the transmission system to reallocate power distribution to other transmission lines connecting the bus, which in turn can trip due to excessive load. This results in a cascading failure and the community connected to the bus can be out of power. This disruption in distribution system generally doesn't stay within a community as from the generation system to the consumers, power flows through transmission lines linked to several buses as a result a total blackout can occur.

In order to simulate this scenario we have used SecAMI [15], an opensource simulator developed to study the impact of cyber-attacks on AMI. SecAMI can be used to perform two operations: first an AMI topology involving smart meters and data concentrators can be created as an undirected graph, second an attack on the created topology can be simulated that compromises one node (smart meter) after the other according to three simulation parameters – compromise time of a node (ct), hop time from one node to another (ht), and detection time of a compromised node (dt). These parameters determine the attack propagation speed with respect to the detection speed in the simulation. We have modified SecAMI to implement the protection-zone aware response mechanism of TRAP. To create protection-zones for a given AMI topology, we have built a partitioning module on top of SecAMI that takes a parameter k and a topology graph as inputs to create k protection-zones of equal sizes. The partitioning module maintains information about the members of a protection-zone and boundary components of the created protection-zones. The response mechanism of TRAP uses the protection-zone information to identify the protection-zones whose member components are detected to be compromised. Subsequently, it immediately

halts the communication of the boundary components of the under attack protection-zones from the rest of the AMI topology.

### 4.6.2 Experimentation and Results

To evaluate the performance of TRAP in terms of damage containment, we have conducted several experiments. The variable parameters in the experiments are: the size of the topology in terms of number of nodes (n), the compromise to detection time ratio ($\sigma = ct/dt$) that specifies the attack propagation speed with respect to detection speed, and the partitioning parameter (k) that specifies the number of protection-zones for a given topology. The performance of TRAP is measured in terms of three metrics. The first two are aforementioned general CPS damage containment evaluation metrics, that are, operational availability and damage extent. The third metric is AEL that is specific to AMI and power grid analysis. In our experiments, we consider attacks on function 1: meter_ops_nan1 i.e. in intrusion boundary 1 in Figure 4.2(b). Since only one function is considered, operational availability is the percentage of components unaffected in the attack and damage extent is the percentage of components compromised in the attack. For AEL we assume that each smart meter in the topology is connected to a medium-sized household with standard appliances like washing machine, dish washer etc. The smart controller of each consumer schedules the energy consumption for next 24h based on the guideline pricing information. The quadratic pricing model presented in [89] is employed to form the normal guideline price and energy consumption for a general household. In addition to evaluating the performance of TRAP in terms of the two damage containment metrics and AEL, we have conducted experiments that illustrate the overhead of response operations of TRAP on the connectivity of CBS components. Whenever an intrusion is detected, the response mechanism blocks all communication from the affected protection-zone to the rest of the CBS. Consequently, the communication of CBS components that are linked directly to the affected protection-zone also becomes restricted. The components with altered connectivity adversely effect the CBS functionality associated with them and the restricted communication of CBS components is quantified through the metric communication impediment.

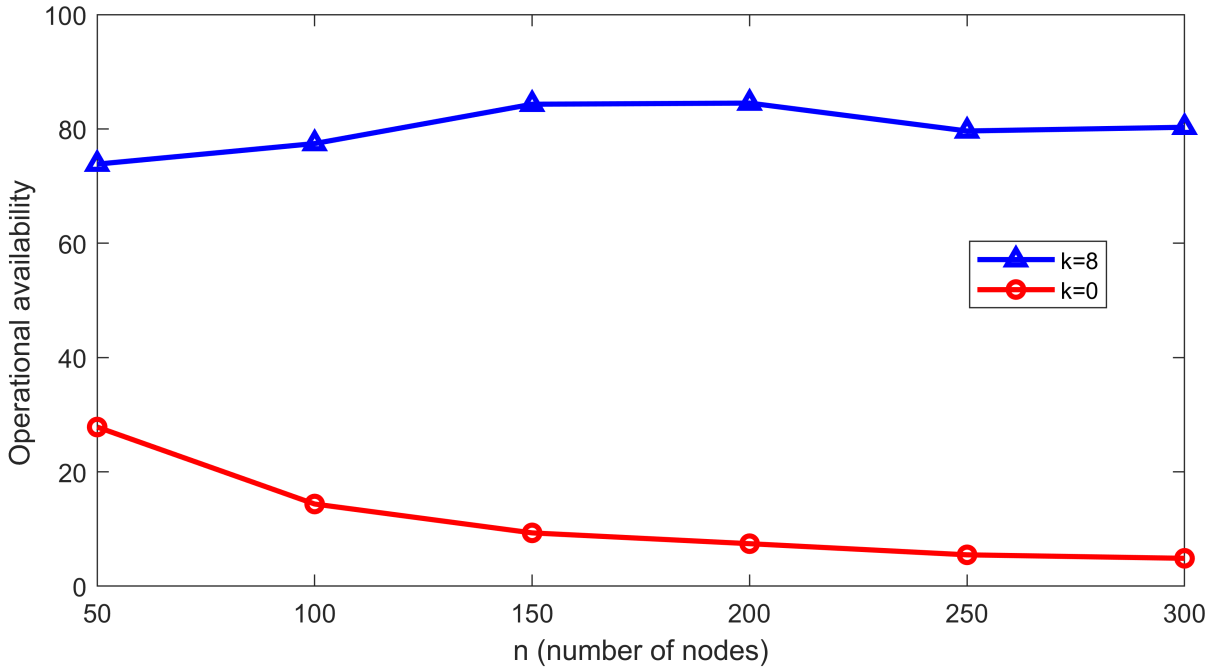


**Figure 4.3.** Operational availability over topology size for $\sigma = 3$.

The value of operational availability for different values of n with $\sigma = 3$ and $k = 0, 8$ is shown in Figure 4.3. It can be seen that the performance of the proposed partition-driven architecture ($k = 8$) is considerably higher as compared to the non-partitioned system ($k = 0$) e.g. at $n = 200$, the value of operational availability is 84.5 for the partitioned system while its value is 7.4 for non-partitioned system. Moreover, note that the value of operational availability increases as the value of n increases until 200 nodes after that it decreases. This is due to the fact that there exists an optimal protection-zone size that maximizes the benefit. If the size of the protection-zone is too small, it is more likely that damage will propagate to other protection-zones. If the size is too large then many member nodes of the protection-zone can get corrupted without any hindrance.

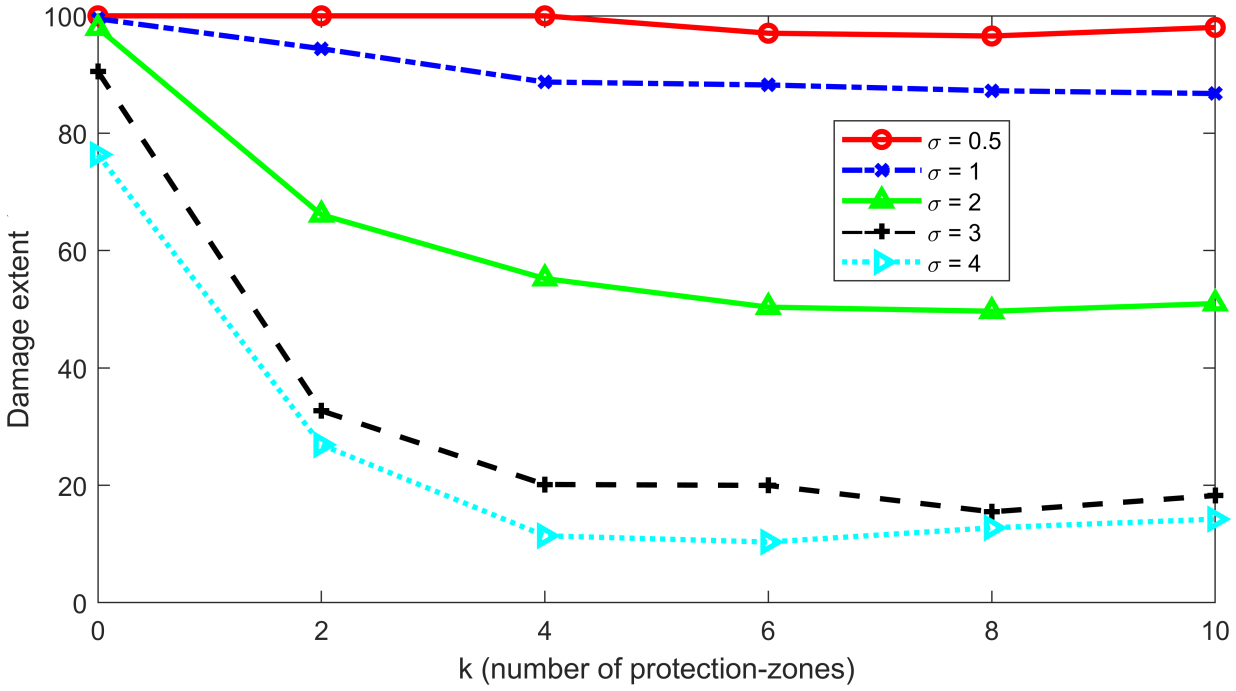


**Figure 4.4.** Damage extent over number of protection-zones for $n = 200$.

The damage extent for a 200 node topology with different $\sigma$ and k values is shown in Figure 4.4. Notice that when the attack propagation speed is very high, that is, lower values of $\sigma$, both the partitioned system ($k > 0$) and the non-partitioned system ($k = 0$) are unable to contain the damage. This is due to the fact that for very fast progressing attacks, when an attack is detected in a certain protection-zone, the attack has already been spread to other protection-zones before the isolation process takes place for an infected protection-zone. This causes even the proactive protection-zone centric response to flounder, though the performance of the proactive response is somewhat better than the reactive detect and disconnect response. For higher $\sigma$ values, the performance of the partition-driven response mechanism is significantly better than the reactive non-partitioned response as there's enough room for the response mechanism to detect the attack in a protection-zone and isolate it before the attack propagates to any other protection-zone.

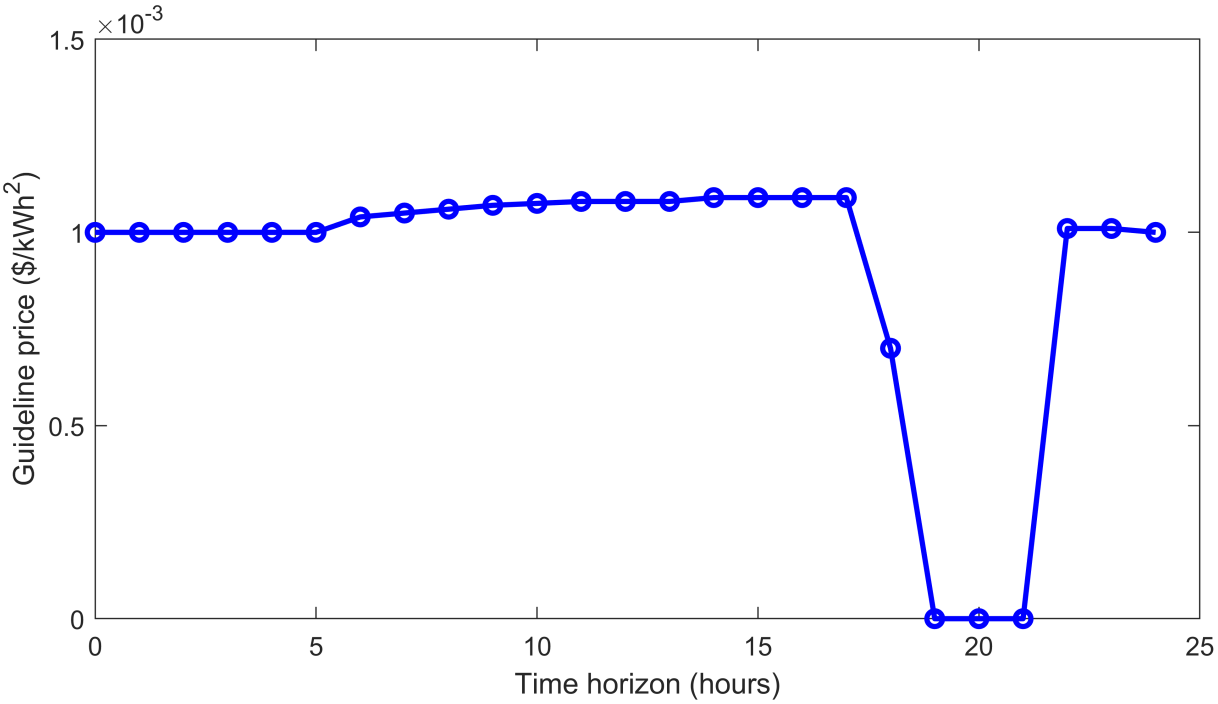


**Figure 4.5.** Guideline electricity price in pricing manipulation cyber-attack.

The manipulated guideline price in a smart meter that is determined using the quadratic pricing model presented in [89] is shown in Figure 4.5. It can be seen that the guideline price drops from $1x10^{-3}$ $\$/kWh^2$ to around $0.7x10^{-3}$ $\$/kWh^2$ at 6PM which further drops to 0 $\$/kWh^2$ from 7PM to 9PM. This change in guideline price increases the energy consumption of a consumer by 1.3 kWh on average in our simulations. The objective of the attacker is to manipulate the guideline price data of as many smart meters as possible so that the energy consumption for many consumers in a neighborhood can be increased. When the attacker is able to change the guideline price data of many smart meters then the overall energy demand of neighborhood is increased. Here in this scenario the attacker is orchestrating a blackout that is initiated at 6PM and ends at 9PM. If the energy demand from a neighborhood increases drastically at a certain time of the day, the transmission line supplying power to that neighborhood becomes vulnerable to malfunction and it is most likely tripped.

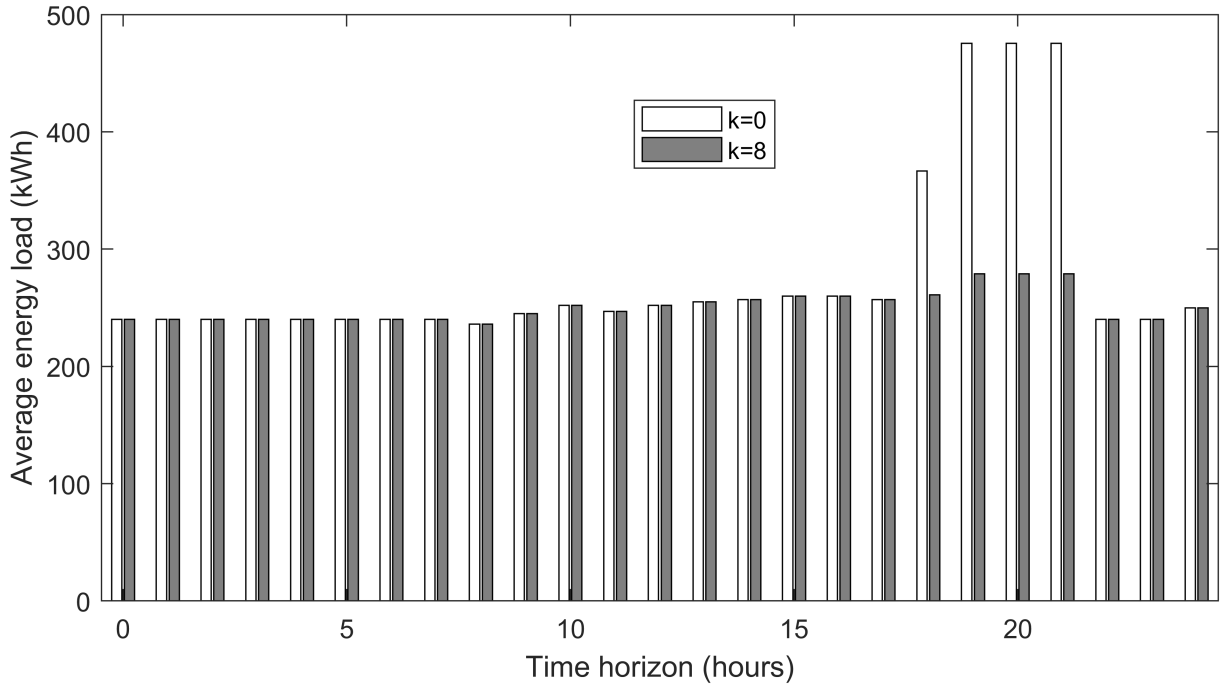


**Figure 4.6.** Overall AEL consumption of a community for $n = 200, \sigma = 3$.

The increase in overall energy demand by all consumers corresponding to the 200 node topology for non-partitioned ($k = 0$) and partitioned systems ($k = 8$) is shown in Figure 4.6. It can be noted that when the pricing manipulation cyber-attack begins at 6PM, the AEL value is observed to be increasing for the non-partitioned system while it remains almost unchanged for the partitioned system. Similarly, between 7PM to 9PM, the AEL increases significantly in the non-partitioned system case while there is just a slight increase in the AEL in the partitioned system case. This demonstrates the effectiveness of the partition-based proactive response mechanism in comparison to the reactive detect and disconnect response mechanism. The partition-driven response mechanism of TRAP efficiently handles the attack and correspondingly load on the transmission lines connected to the bus that connects 200 consumers to the power grid is not drastically increased.

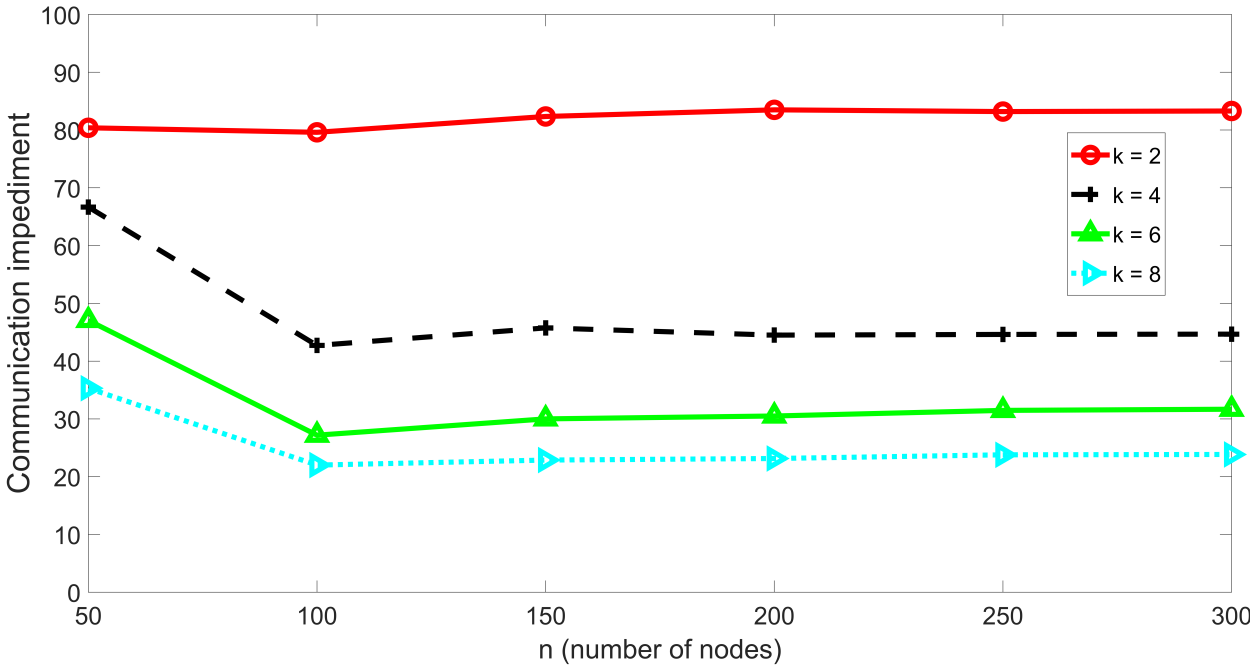


**Figure 4.7.** Communication impediment over topology size for $\sigma = 3$.

The overhead incurred on the connectivity of CBS due to the execution of response operations is shown Figure 4.7. The results are for different values of the topology size $n$ and the number of protection-zones $k$. For $n = 50$, the communication impediment is higher as compared to the other topology size values corresponding to $k = 4$, 5 and 6. This is because when we partition a compact topology into several protection-zones, the sizes of the resultant protection-zones are not large enough to fully contain the attack. Consequently, the attack that starts from one protection-zone propagates to others, and the response mechanism isolates more than one protection-zones. For $n = 50$ and $k = 2$, the sizes of the two protection-zones are large enough to prevent the movement of the attack from one protection-zone to another, accordingly, the values of communication impediment are similar for all topology sizes. For higher values of $n$, the increase in topology size offers the resulting protection-zones enough depth to localize the attack effectively. Additionally, it can be observed that the communication impediment does not vary considerably as the topology size increases because with the increase in topology size, the size of protection-zones also increases. As the communication impediment is the percentage of components with restricted communication corresponding to the total number of components in the CBS, therefore, the

increase in both the protection-zone and topology size does not change the communication impediment significantly. It can be seen that with the increase in number of protection-zones, the percentage of CBS components experiencing restricted communication decreases. This is due to the fact that as the number of protection-zones increases, the protection-zone sizes become smaller for a given topology size. As a result, the communication of fewer number of components is altered when the response mechanism isolates an infected protection-zone. However, as can be seen in results presented in Figure 4.4, if we keep on increasing the number of protection-zones then the damage containment performance starts to degrade after a certain value of k for a given a topology. Therefore, there is a trade off between the damage containment and overhead incurred by the response operations governed by the selection of parameter k. The impact of communication impediment on the functionality of CBS depends on the type of components being subjected to communication restrictions. If the affected components perform multiple functions of vital importance to the CBS then the impact on the overall functionality of CBS will be greater. In the case of AMI, the restricted communication of smart meters can limit the data transmitted back to the electric utility company which can impact the power grid's reliability and efficiency.

# 5. CONCLUSION AND FUTURE WORK

In this chapter, the summary of contributions of this dissertation is presented in section 5.1 and directions of the future research are elaborated in section 5.2.

## 5.1 Summary of Contributions

The contributions of this dissertation are in the addressal of challenges in threat detection and threat response in CBSs. First, to address the challenges in threat detection, the dissertation presents PRISM, a hierarchical intrusion detection architecture for large-scale cyber networks that can predict the progression of complex multi-stage cyber-attacks in realtime by processing a fraction of the total network traffic. To achieve this, PRISM employs an attacker behavior model-based threat-aware sampling scheme and a distributed network traffic monitoring mechanism. Due to its distributed structure, PRISM like every other distributed system, experiences the challenge in alert order preservation that introduces errors in the prediction process. PRISM addresses this challenge by utilizing an efficient alert stream management mechanism. Extensive experimentation has been conducted to evaluate the performance of PRISM under different conditions. It has been shown that PRISM reduces the network traffic processing overhead by up to 750% as compared to a standard IDS while being able to predict the attack progression accurately. PRISM also demonstrates the ability to predict the progression of an attack to advanced stages as soon as the transitions unfold, providing the response infrastructure additional margin to operate in a time constrained environment. Furthermore, multiple security metrics have been proposed that showcase a holistic security outlook of the system to support intrusion response operations. Second, to address the challenges in threat-response, the dissertation introduces TRAP, a partition-driven integrated security architecture that provides a comprehensive security incident handling solution for CBSs. TRAP proposes a novel adaptive partitioning mechanism that localizes the cyber-attack effectively at the region of the cyber-attack's inception. This helps in thwarting fast propagating cyber-attacks and minimizes the damage spread. TRAP also incorporates a recovery mechanism that recovers the affected CBS assets while simultaneously executing response operations. The performance of TRAP has been evaluated by

implementing it in the source code of SecAMI simulator and examining the ability of TRAP in countering a fast propagating cyber-attack.

## 5.2 Future Research Direction

In this section, three future research directions that can be pursued by extending the research presented in this dissertation are discussed.

## 5.3 Security-centric Network Traffic Sampling using Temporal Graph Networks

The threat-aware sampling scheme proposed in this dissertation uses a static graph to model the behavior of an attacker that can be used to determine the devices that are more likely to be targeted in case of a cyber-attack. The static graph is termed as the network reachability graph that represents the devices in the network and their interactions with each other. This static graph model can be extended to a dynamic graph model in which the vertices and edges of the graph evolve with time. A dynamic graph model can yield more accurate results as the nature of a large-scale cyber network is also dynamic. One such model the we plan to explore is known as Temporal Graph Network (TGN) [90]. TGN is a model for deep learning on dynamic graphs that can learn from the evolution of dynamic graphs and then can predict the future interaction among nodes. This can be utilized in threat-aware sampling by training the TGN with attack data and then employing the TGN to predict how network devices can get compromised from other devices in the network.

## 5.4 Multi-stage Attack Prediction using Recurrent Neural Networks

The threat detection architecture presented in this dissertation uses an HMM-based prediction model to assess the progression of a multi-stage cyber-attack. Recently, Recurrent Neural Networks (RNNs)-based prediction models have become popular in scenarios involving time series data centric cyber-attacks. One such RNN-based model we intend to incorporate in our prediction system is Long Short-Term Memory (LSTM) model. We plan to investigate the difference in performance of HMM and LSTM model in their ability to

predict different stages of a complex muli-stage cyber-attack under different experimental conditions.

## 5.5 Dynamic Threat Response using Reinforcement Learning Techniques

The TRAP architecture proposed in the dissertation uses a partitioning mechanism that uses the network topological structure of a CBS to create static partitions according to an optimization problem that essentially minimizes the chances of cyber-attack proliferation into different partitions. The partitions play an important role in response operations by isolating an under attack partition from rest of the CBS. This strategy effectively hampers the progression of a cyber-attack but have limitations as well. One such limitation is the frequent change in the connections among CBS components that can reduce the effectiveness of the partition-based response. Moreover, if the attacker has information about the partitions then this information can be used to circumvent the response mechanism. To address these challenges, the partitioning mechanism can be made dynamic using the reinforcement learning techniques in which for different environmental conditions like changes in the network topology, actions of the attacker etc., the partitions are modified accordingly.

# REFERENCES


[1] M. Biro, A. Mashkoor, J. Sametinger, and R. Seker, "Software safety and security risk mitigation in cyber-physical systems," *IEEE Software*, vol. 35, no. 1, pp. 24–29, 2017.

[2] *Fireeye m-trends 2020 report*. [Online]. Available: https://content.fireeye.com/m-trends/rpt-m-trends-2020.

[3] T. Lukaseder, "Security in high-bandwidth networks," Dissertation, 2020. [Online]. Available: https://oparu.uni-ulm.de/xmlui/handle/123456789/33216.

[4] G. Gilder, *TELESCOM: How infinite bandwidth will revolutionize our world*. Free Press, 2000.

[5] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.

[6] *Zeek cluster architecture*. [Online]. Available: https://docs.zeek.org/en/current/cluster/index.html.

[7] *Snort intrusion detection/prevention system*. [Online]. Available: https://www.snort.org.

[8] *Zeek network security monitor*. [Online]. Available: https://www.zeek.org.

[9] J. Mai, C. Chuah, A. Shridharan, T. Ye, and H. Zang, "Is sampled sufficient for anomaly detection?" In *6th ACM SIGCOMM Conference on Internet Measurement*, Oct. 2006, pp. 165–176.

[10] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security – a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

[11] S. Zonouz, H. Khurana, W. Sanders, and T. Yardley, "Rre: A game-theoretic intrusion response and recovery engine," in *IEEE Transaction on Parallel and Distributed Systems*, vol. 25, 2014, pp. 395–406.

[12] Y. Javed, M. Khayat, A. Elghariani, and A. Ghafoor, "Prism: A hierarchical intrusion detection architecture for large-scale cyber networks," *arXiv preprint arXiv:2111.11000*, 2021. [Online]. Available: https://arxiv.org/abs/2111.11000.

[13] A. Shiravi, H. Shiravi, M. Tavallee, and A. A. Ghorbani, "Towards developing a systematic approach to generate benchmark datasets for intrusion detection," in *Computers & Security*, vol. 31, May 2012, pp. 357–374.

[14] Y. Javed, M. Felemban, T. Shawly, J. Kobes, and A. Ghafoor, "A partition-driven integrated security architecture for cyberphysical systems," *IEEE Computer*, vol. 53, no. 3, pp. 47–56, 2020.

[15] T. Shawly, J. Liu, N. Burrow, and S. Bagchi, "A risk assessment tool for advanced metering infrastructures," in *2014 IEEE Conference on Smart Grid Communications*, 2014.

[16] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," *NIST Inter-Agency Report 7788*, Aug. 2011.

[17] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013.

[18] M. Farmad and A. Bafghi, "Bayesian decision network-based security risk management framework," *Journal of Network and Systems Management*, vol. 28, pp. 1794–1819, 2020.

[19] Y. Lai and P. Hsia, "Using the vulnerability information of computer systems to improve the network security," *Computer Communications*, vol. 30, no. 9, pp. 2032–2047, 2007.

[20] V. Viduto, C. Maple, W. Huang, and D. Perez, "A novel risk assessment and optimization model for a multi-objective network security countermeasure selection problem," *Decision Support Systems*, vol. 53, no. 3, pp. 599–610, 2012.

[21] X. Ou, W. Boyer, and M. McQueen, "A scalable approach to attack graph generation," in *13th ACM Conference on Compter and Communication Security (CCS '06)*, 2006, pp. 336–345.

[22] R. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *13th European Association on Research in Computer Security (ESORICS)*, vol. 5283, 2008, pp. 18–34.

[23] V. Mehta, C. Bartzis, H. Zhu, E. M. Clarke, and J. Wing, "Ranking attack graphs," in *Recent Advances in Intrusion Detection*, Sept. 2006.

[24] R. Dantu, P. Kolan, and J. Cangussu, "Network risk management using attacker profiling," *Security and Communication Networks*, vol. 2, pp. 83–96, 2009.

[25] A. Gusmao, M. Silva, T. Poleto, L. Silva, and A. Costa, "Cybersecurity risk analysis model using fault tree analysis and fuzzy decision theory," *International Jouranl of Information Management*, vol. 43, pp. 248–260, 2018.

[26] M. Barrere, R. Steiner, R. Mohsen, and E. Lupu, "Tracking the bad guys: An efficient forensic methodology to trace multi-step attacks using core attack graphs," in *13th International Conference on Network and Service Management*, Nov. 2017.

[27] D. Malzahn, Z. Birnbaum, and C. Wright-Hamor, "Automated vulnerability testing via executable attack graphs," in *2020 International Conference on Cyber Security and Protection of Digital Services*, June 2020.

[28] G. Androulidakis, V. Chatzgiannikis, and S. Papavassilou, "Network anomaly detection and classification via opportunistic sampling," in *IEEE Network*, vol. 23, Jan. 2009, pp. 6–12.

[29] L. Braun, G. Munz, and G. Carle, "Packet sampling for worm and botnet detection in tcp connections," in *2010 IEEE Network Operations and Management Symposium*, Jun. 2010.

[30] S. Shahreza and Y. Ganjali, "Flexam: Flexible sampling extension for monitoring and security applications in openflow," in *HotSDN 13: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, Aug. 2013, pp. 167–168.

[31] K. Bartos and M. Rehak, "Ifs: Intelligent flow sampling for network security-adaptive approach," *International Journal on Network Management*, vol. 25, pp. 263–282, Jul. 2015.

[32] L. Su, Y. Yao, N. Li, J. Liu, Z. Lu, and B. Liu, "Hierarchical clustering based network traffic data reduction for improving suspicious flow detection," in *12th IEEE International Conference Big Data Science and Engineering*, Sept. 2018.

[33] R. Sibai, Y. Chabchoub, C. Jaoude, J. Demejian, and M. Togbe, "Towards efficient data sampling for temporal anomaly detection in sensor networks," in *IEEE MENACOMM 2019*, Nov. 2019.

[34] D. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

[35] C. Xenakis, C. Panos, and I. Stavrakakis, "A comparative evaluation of intrusion detection architectures for mobile ad hoc networks," *Computers and Security*, vol. 30, pp. 63–80, 2011.

[36] H. Liao, C. Lin, Y. Lin, and K. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, pp. 16–24, 2013.

[37] Z. Ahmad, A. Khan, C. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions Emerging Telecommunications Technologies*, vol. 32, no. 1, Oct. 2020.

[38] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," *IEEE Communication Surveys and Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019.

[39] J. Li, D. Lim, and K. Sollins, "Dependency-based distributed intrusion detection," in *DETER Community Workshop on Cyber Security Experimentation and Test*, 2007.

[40] Y. Zhang, L. Wang, W. Sun, R. Green, and M. Alam, "Distributed intrsuion detection system in a multi-layer network architecture of smart grids," *IEEE Transactions on Smart Grid*, vol. 2, 2011.

[41] C. Gruhl, F. Beer, H. Heck, B. Sick, U. Buhler, A. Wacker, and S. Tomforde, "A concept for intelligent collaborative network intrusion detection," in *ARCS*, 2017.

[42] Y. Xie, Y. Wang, H. He, Y. Xiang, S. Yu, and X. Liu, "A general framework for modeling and perceiving distributed network behavior," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3162–3176, 2016.

[43] O. Alkadi, N. Moustafa, B. Turnbull, and K. Choo, "A deep blockchain framework-enabled collaborative intrusion detection for protecting iot and cloud networks," *IEEE Internet of Things Journal*, May 2020.

[44] M. Husak, J. Komarkova, E. Bou-Harb, and P. Celeda, "Survey of attack projection, prediction and forecasting in cyber security," *IEEE Communication Surveys and Tutorials*, vol. 21, no. 1, pp. 640–660, 2019.

[45] D. S. Fava, S. R. Byers, and S. J. Yang, "Projecting cyberattacks through variable-length markov models," *IEEE Transactions on Information Forensics and Security*, vol. 3, pp. 359–369, 2008.

[46] H. Du, D. F. Liu, J. Holsopple, and S. J. Yang, "Toward ensemble characterization and projection of multistage cyber attacks," *International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[47] F.Manganiello, M. Marchetti, and M. Colajanni, "Multistep attack detection and alert correlation in intrusion detection systems," *International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[48] B. Zhu and A. Ghorbani, "Alert correlation for extracting attack strategies," *International Journal of Network Security*, vol. 3, pp. 244–258, 2006.

[49] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer Communications*, vol. 29, pp. 2917–2933, 2006.

[50] X. Cheng and S. Lang, "Research on network security situation assessment and prediction," in *2012 Fourth international conference on computational and information sciences*, Aug. 2012.

[51] L. Shang, W. Zhao, J. Zhang, Q. Fu, Q. Zhao, and Y. Yang, "Network security situation prediction based on long short-term memory network," in *2019 20th Asia-Pacific network operations and management sysmposium*, Nov. 2019.

[52] O. Fredj, A. Mihoub, M. Krichen, O. Cheikhrouhou, and A. Derhab, "Cybersecurity attack prediction: A deep learning approach," in *SIN 2020: 13th international conference on security of information and networks*, Nov. 2020, pp. 1–6.

[53] H. Yang, R. Zeng, G. Xu, and L. Zhang, "A network security situation assessment method based on adversarial deep learning," *Applied Soft Computing*, vol. 102, no. 12, 2021.

[54] S. Cho, "Incorporating soft computing techniques into a probabilistic intrusion detection system," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 32, no. 2, pp. 154–160, 2002.

[55] S. Joshi and V. Phoha, "Investigating hidden markov models capabilities in anomaly detection," in *ACM-SE 43: 43rd Annual Southeast Regional Conference*, Mar. 2005, pp. 98–103.

[56] G. Florez-Larrahondo, S. Bridges, and R. Vaughn, "Efficient modeling of discrete events for anomaly detection using hidden markov models," in *ISC'05: 8th International Conference on Information Security*, Sept. 2005, pp. 506–514.

[57] S. Cho, "Hmmpayl: An intrusion detection system based on hidden markov models," *Computer and Security*, vol. 30, no. 4, pp. 221–241, 2011.

[58] N. Gornitz, M. Braun, and M. Kloft, "Hidden markov anomaly detection," in *32nd International Conference on Machine Learning*, 2015, pp. 1833–1842.

[59] K. Haslum, M. Moe, and S. Knapskog, "Hidden markov anomaly detection," in *International Conference Local Computer Network*, 2008.

[60] H. Farhadi, M. AmirHaeri, and M. Khansari, "Alert correlation and prediction using data mining and hmm," *ISC International Journal Information Security*, vol. 2, no. 2, pp. 77–101, 2011.

[61] A. Sendi, M. Dagenais, M. Jabbarifar, and M. Couture, "Real time intrusion prediction based on optimized alerts with hidden markov model," *Journal of Networks*, vol. 7, no. 2, pp. 311–321, 2012.

[62] P. Holgado, V. Villagra, and L. Vasquez, "Real-time multistep attack prediction based on hidden markov models," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 134–147, 2017.

[63] T. Shawly, A. Elghariani, J. Kobes, and A. Ghafoor, "Architectures for detecting interleaved multi-stage network attacks using hidden markov models," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[64] T. Chadza, K. Kyriakopoulos, and S. Lambotharan, "Learning to learn sequential network attacks using hidden markov models," *IEEE Access*, vol. 8, pp. 134 480–134 497, 2020.

[65] A. Sendi, N. Jivan, M. Jabbarfar, and M. Dagenais, "Intrusion response systems: Survey and taxonomy," *International Journal of Computer Science and Network Security*, vol. 12, 2013.

[66] P. E. Verissimo, N. F. Neves, and M. P. Correia, "Intrusion-tolerant architectures: Concepts and design," in *Lecture Notes in Computer Science*, vol. 2677, 2003.

[67] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for intrusion detection and response," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, 2000.

[68] X. Wang, D. Reeves, S. Wu, and J. Yuill, "Sleepy watermark tracing: An active network-based intrusion response framework," in *IFIP International Information Security Conference*, 2001, pp. 369–384.

[69] S. Lannuci and S. Abdelwahed, "Model-based response planning strategies for autonomic intrusion protection," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 13, no. 1, pp. 1–23, May 2018.

[70] E. Miehling, M. Rasouli, and D. Teneketzis, "A pomdp approach to the dynamic defense of large-scale cyber networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2490–2505, Oct. 2018.

[71] E. Jonsson and T. Olovsson, "A quantitative model of the security intrusion process on attacker behavior," *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 235–245, 1997.

[72] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 4, pp. 107–117, 1998.

[73] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[74] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernandez-Moctezuma, R. Lax, S. McVeety, F. P. D. Mills, E. Schmidt, and S. Whittle, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of VLDB Endowment*, vol. 8, no. 12, 2015.

[75] *Common vulnerability scoring system*. [Online]. Available: https://www.first.org/cvss/.

[76] *Nessus network vulnerability scanner*. [Online]. Available: https://www.tenable.com/products/nessus/.

[77] *Openvas - open vulnerability assessment scanner*. [Online]. Available: https://www.openvas.org/.

[78] *Nmap: The network mapper*. [Online]. Available: https://nmap.org/.

[79] U. Srivastava and J. Widom, "Flexible time management in data stream systems," in *Proc. ACM PODS 2004*, Jun. 2004, pp. 263–274.

[80] *Mitre att&ck*. [Online]. Available: https://attack.mitre.org/.

[81] *Splunk enterprise security*. [Online]. Available: https://www.splunk.com/.

[82] *Extrahop*. [Online]. Available: https://www.extrahop.com/.

[83] *Logrhythm*. [Online]. Available: https://logrhythm.com/.

[84] D. Jurasky and J. Martin, *Speech and language processing*. Pearson Prentice Hall, 2009.

[85] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *ACM SIGCOMM IMW'02*, Novemeber 2002.

[86] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[87] P. Liu, "Architectures for intrusion tolerant database systems," in *18th Annual Computer Security Conference*, 2002.

[88] Y. Liu, S. Hu, and A. Zomaya, "The hierarchical smart home cyberattack detection considering power overloading and frequency disturbance," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1973–1983, 2016.

[89] L. Liu, Y. Liu, L. Wang, S. Hu, and A. Zomaya, "Economical and balanced energy usage in the smart home infrastructure: A tutorial and new results," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 4, pp. 556–570, 2015.

[90] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020. [Online]. Available: https://arxiv.org/abs/2006.10637.

# VITA

Yahya Javed completed his Bachelor of Engineering degree in Information and Communication Systems from the School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan in 2013. Before joining the grad school at Purdue, Yahya worked as a research assistant at NUST, PTCL & Cisco Center of Excellence for IP Technologies, Islamabad, Pakistan. Yahya began his Ph.D. in Computer Engineering at the Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA in 2015. Yahya received the 2009-2013 Shahen merit scholarship and NUST academic excellence scholarship in 2011-2013. Yahya was the recipient of VIP Graduate Mentor Fellowship in 2021.