A Game-Theoretic Decision-Making Framework for Engineering Self-Protecting Software Systems

by

Mahsa Emami-Taba

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Dr. Mohammad Zulkernine Professor and Canada Research Chair, School of Computing, Queen's University
Supervisor:	Dr. Ladan Tahvildari Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Dr. Otman Basir Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
	Dr. Vijay Ganesh Associate Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Dr. Reid Holmes Adjunct Associate Professor, Cheriton School of Computer Science, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Targeted and destructive nature of strategies used by attackers to break down a software system require mitigation approaches with dynamic awareness. Making a right decision, when facing today's sophisticated and dynamic attacks, is one of the most challenging aspects of engineering self-protecting software systems. The challenge is due to: (i) the consideration of the satisfaction of various security and non-security quality goals and their inherit conflicts with each other when selecting a countermeasure, (ii) the proactive and dynamic nature of these security attacks which make their detection and consequently their mitigation challenging, and (iii) the incorporation of uncertainties such as the intention and strategy of the adversary to attack the software system.

These factors motivated the need for a decision-making engine that facilitates adaptive security from a holistic view of the software system and the attacker. Inspired by game theory, in this research work, we model the interactions between the attacker and the software system as a two-player game. Using game-theoretic techniques, the self-protecting software systems is able to: (i) fuse the strategies of attackers into the decision-making model, and (ii) refine the strategies in dynamic attack scenarios by utilizing what has learned from the system's and adversary's interactions.

This PhD research devises a novel framework with three phases: (i) modeling quality/malicious goals aiming at quantifying them into the decision-making engine, (ii) designing game-theoretic techniques which build the decision model based on the satisfaction level of quality/malicious goals, and (iii) realizing the decision-making engine in a working software system. The framework aims at exhibiting a plug-and-play capability to adapt a game-theoretic technique that suite security goals and requirements of the software. In order to illustrate the plug-and-play capability of our proposed framework, we have designed and developed three decision-making engines. Each engine aims at addressing a different challenge in adaptive security. Hence, three distinct techniques are designed: (i) incentive-based ("IBSP"), (ii) learning-based ("MARGIN"), and (iii) uncertainty-based ("UBSP"). For each engine a game-theoretic approach is taken considering the security requirements and the input information. IBSP maps the quality goals and the incentives of the attacker to the interdependencies among defense and attack strategies. MARGIN, protects the software system against dynamic strategies of attacker. UBSP, handles adversary-type uncertainty. The evaluations of these game-theoretic approaches show the benefits of the proposed framework in terms of satisfaction of security and non-security goals of the software system.

Acknowledgements

First and foremost I would like to thank my enthusiastic supervisor Professor Ladan Tahvildari. It was under Dr. Tahvildari's supervision that I prospered in this compelling research area, and ultimately, was able to meet and pass all the milestones one after another. My Ph.D. journey has been an amazing experience, not only for her tremendous academic support, but also for having her as a mentor and a sister. In addition to our academic collaboration, I greatly value the close personal rapport that Ladan and I have forged over the years. Words cannot express my gratitude for all her sweet gestures, encouragements and kind support during my ups and downs of starting a new life in Waterloo.

I would like to thank the members of my dissertation committee. I was so fortunate to have such an intellectual committee with me along the path: Professor Mohammad Zulkernine my external examiner, for having accepted to take the time out of his busy schedule to read my thesis and provide me with his insightful suggestions, Dr. Otman Basir for his encouraging comments and indispensable support, Dr. Vijay Ganesh for his inspiring remarks which broadened my perspectives in research, Dr. Reid Holmes, my internal-external examiner for his invaluable comments on my thesis, and Dr. Adam Kolkiewicz for serving as the thesis examination chair.

I am grateful to have had the opportunity to meet Professor Hausi Muller. I cannot thank him enough for all his valuable feedback and kind support. I am particularly indebted to Dr. Marin Litoiu for exposing me to their testbed infrastructure at CERAS lab. I was fortunate to be part of the collaboration project with Professor Ettore Merlo and Dr. Marios-Eleftherios Fokaefs from Ecole Polytechnique de Montreal, Viorel Onut from IBM Markham, and CERAS lab member Nasim Beigi Mohammadi. Also, I would like to express my gratitude to Dr. Patrick Martin for his valuable feedback and remarks during the ORF-RE project on Ultra-Large-Scale Services meetings.

I would like to express my sincere appreciation to Communications Security Establishment Canada (CSEC) to open my horizon to do this line of research. I would like to take time and thank the Doctoral Symposium committee, and the ACM student research competition committee, at the International Conference on Software Engineering (ICSE)-2017, whose constructive comments, and influential criticisms, paved the way for accomplishing my research goals.

I am very lucky to have been given the opportunity to be a part of Lovell Corporation. I am grateful for their contribution in my academic life as a start up company to see where some techniques used in my thesis can have a direct application in a

young company. Many thanks to Kelly Lovell for providing a pleasant and friendly working and research environment.

I would like to thank my colleagues in the Software Technologies and Applied Research (STAR) lab, Dr. Mohammad Hamdaqa and Soheil Soltani for their moral support and valuable feedback. In particular, I would like to thank Dr. Mehdi Amoui for all the brainstorming sessions and discussions.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by Natural Sciences and Engineering Research Council (NSERC) Engage and Ontario Centres of Excellence (OCE) Voucher for Innovation and Productivity (VIP), University of Waterloo Provost Doctoral Entrance Award, Ontario Graduate Scholarship (OGS), University of Waterloo Presidents Graduate Scholarship (PGS), and Lovell Corporation.

Finally, but by no means least, I owe an immense debt of gratitude to my awesome Mom and Dad, who raised me with love and supported me in all my pursuits, and to my sisters, Leila and Parisa, who were always next to me. Thank you for all the love and support and sacrifice you have made through bad times and for being a part of my happiness in the good times. This last word of acknowledgment I have saved for my caring and encouraging friend, Abbas, whose faithful support during the final stages of this Ph.D. is so appreciated.

Dedication

To my beloved parents and sisters.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today's life style is largely dependent on software systems. Securing these systems and their information are a critical concern from personal to governmental scale standpoints. However, the increasing complexity of software systems hardens the achievement of the desired level of security. In these systems, a great challenge for administrators is to select a timely countermeasures in response to attacks. Nonetheless, the dynamic nature of security attacks demands fast reacting adaptive systems that are able to detect and mitigate threats on the fly while ensuring the security goals of Confidentiality, Integrity, and Availability (CIA) [1]. Designing Self-Protecting Software (SPS) systems is a response to these demands.

The main objective of SPS systems is to satisfy security goals and requirements. An important issue in these systems is to make adaptation decisions as a response to threats or to the deviations from security goals. The aforementioned issue has set the main goal of this PhD research thesis. More specifically, we aim at providing a novel systematic approach for decision making in SPS systems.

1.1 Motivation

The number of application-layer attacks (such as "low and slow" Distributed Denial of Service (DDoS) attacks) is rising in the past few years[2]. According to the Arbor's

[1] https://en.wikipedia.org/wiki/Information_security/

[2] https://www.arbornetworks.com/blog/insight/application-layer-ddos-attacks-the-numbers-may-surprise-you/

12th annual Worldwide Infrastructure Security Report (WISR), the average cost of downtime to the victims of a DDoS attack is around $500 per minute[3]. The report also indicates that 25% of DDoS attacks target applications which means about 2 million attacks annually. The challenge is that malicious requests can look like legitimate requests until the application can no longer respond.

Incapsula[4], a company that provides website security and DDoS protection services, announced the mitigation of one of the most sophisticated and highly adaptive DDoS attacks they have faced so far[5]. The attack was against one of its customers that was a popular trading site and was suspicious to be initiated from the ex-partner of the targeted company as the attacker has intimate knowledge of the vulnerabilities in the targeted infrastructure. The attack was ongoing for few weeks until it was fully mitigated. It was a combination of various network-layer and application-layer attacks. The unique characteristic of this attack was the dynamic change of attack type based on the defense strategy that is taken by Incapsula. For example, it started with a network *SYN flood attack* and was followed by various application-layer attacks such as *HTTP flood attack* that targeted several chosen resource intensive pages. These attempts were mitigated by approaches such as sufficient *network capacity* and *client classification*. However, interestingly the attacker seems to be able to observe failure of those attacks and started another type of application-layer attack by targeting the *AJAX objects* of website which results in severe impact on the database.

As Incapsula kept blocking the different attack methods, the attacker kept adapting. Finally, the attacker started flooding the website with requests that appear to be legitimate as they were sent from real browsers using *malware-infected computers*. One of the Incapsula researchers said: "It looked like an abnormally high spike in human traffic". Incapsula researchers were able to *manually* detect this attack and apply *CAPTCHA challenges* to mitigate it.

As highlighted above, software systems or software security providers, cannot rely on rule-based, policy-based, or goal-based decision-making approaches [7][99]. These approaches either fail to protect the software system when facing dynamic attacks or become too complicated to capture and maintain all possible scenarios. Attack mitigation is often hard coded in these applications. As a result, responding to uncertain types of attacks is not possible. Moreover, intrusion response at the

[3]https://www.arbornetworks.com/blog/insight/ddos-attacks-2017-no-days-off/

[4]http://www.incapsula.com/

[5]https://www.pcworld.com/article/2056805/applicationlayer-ddos-attacks-are-becoming-increasingly-sophisticated.html

later stages of an attack results in more harm to the system in terms of cost and annoyance of legitimate users. Hence, a great challenge in engineering SPS systems is to design a decision-making engine that is capable of selecting a timely countermeasure. Providing this capability, calls for SPS systems with innovative and adaptive decision-making engines that have a set of countermeasures in place to compete with well-planned strategies of attackers.

Furthermore, adaptive application security involves making decisions under uncertainties such as the time, the power, or the damage of potential attacks. One of the uncertainties that has been largely ignored in the literature is the intention of the adversaries. The majority of research focuses on characteristics of *attacks* (e.g., their request arrival rates), whereas characteristics of *attackers/adversaries* (e.g., their intentions and strategies) are neglected. For example in case of Incapsula, the attack was initiated from an ex-partner whom has more knowledge of the application than a malicious user. In today's sophisticated attacks, in order to confuse defense systems, adversaries may initiate an attack that exhibits a scenario similar to another attack but has an entirely different malicious goal (e.g., to break down the server or to harm a specific user in the system). In such cases, incorporating uncertainty about the type of adversaries into the decision model helps to choose a proper countermeasure for protecting the software systems.

1.2 Problem Description and Research Focus

In this research work, capturing the inherent *interdependencies* between strategies of the SPS system and the attacker is the main concern in designing the decision model. The success or failure of an attack depends on how the system is protected, and similarly the effectiveness of a countermeasure depends on how the system is being attacked. In a nutshell, there is a need for decision models that can be adapted based on the strategies of attackers. To address this gap, we employ game theory in order to quantitatively analyze multi-players actions and utilities in SPS.

Problem Statement: *How to model, design, and realize a novel decision-making engine in SPS systems with the aid of game-theoretic techniques?*

To employ a game-theory technique, we first need to model the payoffs of the players (SPS and the attacker). The objective is to incorporate the knowledge gathered from the goals and strategies of the players. Modeling the goals of the players facilitates interpretation of the relation among various sources of data. Goal models

help to capture the positive or negative impact of strategies and hence defining the payoffs.

In this research, the focus is to model and incorporate goals and strategies of attackers into the deciding-making engine. The inputs to this engine are the sets of goals and strategies which are defined for both the SPS system and the attacker. The output of the decision-making engine is a countermeasure representing the chosen strategy by the SPS system while taking into consideration the strategy of the attacker.

1.3 Approach and Research Questions

In our research approach, we aim at engineering a novel decision-making framework that utilizes game theoretic techniques to select the proper mitigation against an attack. The framework will consist of three phases including: (i) modeling quality goal, (ii) designing game-theoretic techniques, and (iii) realizing the decision-making engine. The first phase models the security goals of the system and afterwards maps the goal-oriented model to the designed game-theoretic technique. The goal-oriented model empowers the decision-making engine capable of tracking the satisfaction of the goals before and after applying a mitigation strategy. The framework provides the steps to map the goal-oriented model to the employed game-theoretic technique. The technique is chosen by taking into consideration the available information from the attack scenario and the security requirements of the software system.

The approach aims at providing answers to four research questions. For each of these questions, we explain the objective and the followed methodology,

1.3.1 RQ1: How should the framework incorporate the incentives of the attacker?

Objective: Considering the intention behind an attack facilitates mitigation of both dynamic and uncertain attacks. We noticed that one of the shortcomings of the traditional approaches is their only concentration on the modeling of the security goals, while modeling the malicious goals of attackers is over-sighted.

Methodology: To incorporate the strategies and incentive of the attacker, we model the malicious goals of the attacker. The possible malicious goals can also be determined by investigating the history of attacks to the software system. We start from

defining the high-level goals, then we break them to lower-level goals that can be related to measurable attributes. Chapter 4 provides the details of incorporating the incentive and strategies of attackers.

1.3.2 RQ2: How should the framework learn from previous action selections?

Objective: Learning from the impact of previous countermeasures guides the decision-making engine to take more proper countermeasures in future action selections. Such information will guide the decision-making engine to make better decisions specifically when facing dynamic attacks.

Methodology: To provide a feedback loop in the decision model, we quantify the positive/negative impact of the previously applied countermeasures and incorporate this information into the decision model. Chapter 5 provides the details of incorporating the feedback from previous action selections into the decision model.

1.3.3 RQ3: How should the framework support adversary-type uncertainty?

Objective: Considering uncertainty about the type of the adversary facilitates mitigation of the malicious user behavior as early as possible (before the type of the attack is detected). When an attack scenario is similar among various adversary-types, there is a need to model the characteristics of adversaries/attackers instead of merely modeling the characteristics of attacks.

Methodology: To equip the proposed framework to support adversary-type uncertainty, we consider probabilistic approaches that can define uncertainty in utility values. The details of the proposed Bayesian game decision-making model is provided in Chapter 6.

1.4 Research Contributions

The objectives of our research to the problem of countermeasure selection in SPS systems is engineering a framework that considers goals and strategies of the system and malicious users in its decision-making engine. The contributions of this research are:

- **a novel mapping from high-level quality/malicious goals to quantitative utility values**. The proposed framework allows reasoning about security goals considering other quality goals such as performance, cost, or usability depending on the business goals of the system and hence capturing the trade-offs among security and non-security goals. The modeled framework not only considers the quality goals regarding the adaptable system, but also incorporates the malicious goals of attackers as the basis for reasoning in choosing a countermeasure. Thus, satisfaction of these goals need to be mapped to utility values to make action selection possible.

- **a unique modeling of the interdependencies among defense and attack strategies**. Whether an attack can succeed relies on how the system is protected. Similarly, a defense strategy is effective depending on how the system is attacked. The proposed framework captures such interdependencies among strategies.

- **a learning-based decision model during runtime**. The proposed framework not only mitigates the fixed-strategy attacks, but also prevents dynamic attacks. Such attacks change their strategies in order to stay hidden from the intrusion detection system.

- **an innovative decision model for adversary-type uncertainty**. If the type of the adversary is unknown, the mitigation have to be postponed until the intention of the attacker is clear which happens at the later stages of the attack. Hence, mitigating such attack scenarios can be extremely costly. The proposed framework addresses the scenarios when the software system is uncertain about the type of the adversary as soon as the malicious behavior is detected.

- **a step-wise plug-and-play framework**. The proposed framework provides the guidelines that can be employed to engineer a self-protecting software. The guidelines are generic enough that does not require specific system requirements. Hence, various techniques can be employed depending on the available information as well as the security requirements of the software system.

The research achievements can be enumerated as follows:

• The idea of a holistic and game-theoretic decision-making for adaptive security was discussed in [43]. This article was later selected to be published in the Cyber Security E-Book: *Best of TIM Review, Kindle Edition.*

• The modeling aspect of self-protecting software systems in our research methodology was presented in the Canadian Consortium for Software Engineering Research (CSER) in November 2014. The poster received the *Best Poster* award (3rd place).

• A study in mitigating application-layer SIP flood attack in VoIP telephony systems is conducted by applying the Markov game technique and the obtained results were presented in the 24th IBM Annual International Conference on Computer Science and Software Engineering. The presented poster won the *IBM Best Exhibit* award.

• Another article discussed the idea of adaptation based on the strategy of the attacker and mitigation of attacks considering preferences in quality goals. This position paper was published in the Emerging Technologies Track (ETT) of the 24th IBM Annual International Conference on Computer Science and Software Engineering [44].

• Our paper published in [45] elaborates on our study of strategy-aware mitigation using Markov games for dynamic application-layer attacks. In this paper, a simulated case study using MATLAB Simulink was performed to gauge our game-theoretical approach in SPS systems.

• Our paper published in [46] focuses on addressing uncertainty about adversary types in SPS systems using Bayesian game technique. This paper won the *Best Student Paper* award in the 26th IBM Annual International Conference on Computer Science and Software Engineering.

• The idea of my thesis is described in [42] which was discussed at the Doctoral Symposium of the 39th Annual International Conference on Software Engineering (ICSE).

• A poster was presented in the 39th Annual International Conference on Software Engineering (ICSE) elaborating on the idea of game-theoretic approach for decision-making in SPS [41].

• The idea of strategy-aware mitigation using Markov games [45] is extended as a journal draft and has been submitted to ACM Transactions on Autonomous and Adaptive Systems (TAAS) journal. Currently, the submission is awaiting reviewer scores.

• The position paper that was published in [44] is extended as a full journal draft and is under its final revision to be submitted to Journal of Systems and Software (JSS).

1.5 Document Organization

The rest of this thesis is organized as follows. Chapter 2 briefly reviews the background concepts and some related works. Chapter 3 details the proposed plug-and-play framework. Chapter 4 shows how the framework has been realized to incorporate the incentive of the attacker. Chapter 5 extends the framework to support learning from previous action selection. Chapter 6 shows how to expand the framework to integrate adversary-type uncertainty. Chapter 7 summarizes the thesis, draws several conclusions, and suggests ideas for potential future works.

Chapter 2

Background and Related Works

In recent years, interests in building software systems that are adaptive to their security goals has increased. Self-Adaptive Software (SAS) systems address automation in response to changes in the requirement and environment. SAS *monitors* itself and its context, *detects* significant changes, *decides* how to react, and *executes* such decisions [109]. SAS systems monitor the software *itself* and/or its *context* based on the specific goal they are intended to fulfil. In these systems, the software behavior is evaluated at runtime and pre-defined actions will be executed in case of detecting anomaly in the bahavior of software. The objective of these systems is to ease the need for continuous human supervision. The need for a self-adaptive software was recognized by DARPA in 1997. The Broad Agency Announcement (BAA) [73] describes Self Adaptive Software as: "*Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.*" As software systems become more complicated and diverse, anticipating the interactions among components by software architects are less possible, which leads to software systems dealing with these issues at runtime. Even skilled humans are not able to integrate such complex and massive software systems.

Installing, configuring, optimizing, and maintaining a software system at runtime is a challenge that first was addressed in 2001 by IBM. The only option remaining to tackle the aforementioned problem is autonomic computing. The inspiration of autonomic computing comes from biological systems. For example, heart rate is administered by nervous system unconsciously. It can get slower or faster depending on different conditions without requiring brain instruction. Biological systems cooperate with complexity, heterogeneity and uncertainty [107]. Our conscious brain

does not need to worry about vital and lower-level decision makings as autonomic nervous system deals with these functions. The idea of autonomic computing is coming from the same perspective. It frees system administrators in dealing with lower lever decision makings in management and tedious operational tasks. Here, we briefly name the 8 elements of autonomic computing recognized by IBM [32].

1. *Self-Aware* is the capability of the systems to be aware of itself by means of its states and behaviours. It is provided by the system monitoring itself.

2. *Context-Aware* is the ability of the system to be aware of its operational environment and its changes.

3. *Self-Configuring* is the capability of installing, configuring, and integrating large complex systems. In these systems, a new component will incorporate and adopt itself seamlessly.

4. *Self-Optimizing* finds ways to improve their performance, response time and throughput. In a self-optimizing system, low priority tasks can use computing resources when computing resources are not fully used by higher priority tasks.

5. *Self-Healing* is the capability of recovering from anomalous behaviour. Self-healing is the process of detecting, diagnosing, and repairing localized problems resulting from bugs or failures in software and hardware.

6. ***Self-Protecting*** is the capability of detecting security attacks and triggering countermeasures. These systems not only defend against the malicious attack but also are capable of anticipating problems and taking steps to avoid them or moderate their effects.

7. *Anticipation* is an optional property. An autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden from the user.

8. *Openness* is an optional property. Computing resources are product of different vendors that may or may not share the detail of their product. Managing these computing resources is a challenge since the focus is on building tools rather than on automation of decision making.

IBM [64] outlines existing and emerging standards related to autonomic managers as well as sensors and effectors. Two comprehensive roadmaps on software engineering for self-adaptive systems published in 2009 [27] and 2013 [77] covering a different

set of topics and challenges that must be addressed by this community. To ensure that user requirements are satisfied and quality attributes met their expectations, software Validation and Verification (V&V) methods are developed. Developing certified runtime V&V mechanisms in self-adaptive systems is discussed further in [122].

Self-Protecting Software (SPS) systems are a class of self-adaptive software systems that aim at protecting system against malicious attacks and accidental cascading of failures. SPS systems are developed using existing self-adaptive software development methods aiming at reducing human involvement in protecting systems against malicious attacks. Automation accelerates the analyses of the monitored data and perhaps increases the number of symptoms that can be detected in order to prevent security threats. Moreover, automation helps to speed up the decision-making process at the time of the attack. In an SPS system an immediate, suboptimal response can sometimes be more effective than a late, optimal response. These timely actions prevent the spread of attacks and therefore minimize the consequences of the attacks.

2.1 Self-Protecting Software Principles and Requirements

An SPS needs to anticipate threats based on early reports from sensors and then take steps to prevent or alleviate them. The goal of a SPS is to recognize and deal with attacks without human administrators being involved in the decision making. SPS systems have the ability to detect security attacks and trigger countermeasures. These systems not only defend against malicious attacks but also are capable of anticipating problems and taking steps to avoid them or moderate their effects [109]. Adaptation engine in SPS as well as SAS can be realized using *internal* or *external* approaches [109]:

- *Internal* approaches fuse software and adaptation specification while external approaches use an external adaptation engine. Internal approaches suffer from costly maintenance and lack of scalability because of low level adaptation mechanism they use (e.g., programming-based mechanism).
- *External* approaches are reusable, scalable, and easily maintainable.

In SPS systems that are based on application-layer information, integrating adaptation engine can be either internal or external to the software system. At the network

layer, this component is mostly external because sensors relay on the data coming from the network to the software system as opposed to the data transferred inside the system.

In an autonomous system, autonomic manager's functionality is organized into four groups of functions called MAPE-K (Monitoring, Analyzing, Planning and Executing) loop. These processes share a common *Knowledge* of the system (Figure 2.1).

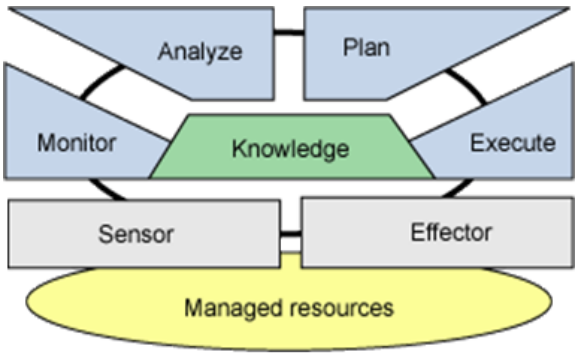


Figure 2.1: IBM MAPE-K Loop [33]

• **Monitor (Observe):** Monitoring is fundamentally important in supporting autonomous systems. It collects data from a managed resource (software system) or the software environment through sensors. The collected data can be aggregated, correlated or filtered if necessary. The data gets ready to be analyzed further. Collected data includes information regarding system status, throughput or etc. For instance, log files contain information about severity of the error and its trace back. Monitoring can also be achieved through heartbeat mechanism. During runtime, it is important that monitoring functions quickly collect and organize the data and provide them to the analyzer functions.

• **Analyze (Detect):** A self-protecting system can detect illegal activities. It analyzes the changes to ensure business goals are being met. Otherwise, the adaptation engine need to plan to adapt the system in order to meet the system policies. To analyze complex behaviors of the system, modeling mechanisms aid autonomic manager not only to analyze the managed resource (or its IT environment) but also to predict the future behavior of the managed resource with the aid of modeling techniques such as time-series forecasting or queuing models. Intrusion Detection Systems (IDSs) compare data packets with a library of known attacks.

• **Plan (Decide):** Planning is to construct a plan of actions in order to meet the objectives and business goals. The resulted plan needs to be aligned with the

defined policies. Planning can be of different granularity such as running a function or setting up a new server. Intrusion prevention systems plan on how to react to the detected malicious attacks. Self-adaptation requires two kinds of plannings, observation planning and adaptation planning. The former is planning what observations are required and the frequency of the observations and their cost. The latter is to determine which adaptation to make, when and where to make the adaptation.

• **Execute (Act):** Execute functionality performs the necessary changes in the provided plan through effectors. Execution of the changes can involve updating the knowledge that is used by the autonomic manager [64]. This process is responsible for applying security function at runtime.

• **Knowledge:** Knowledge contains particular data that can be shared between MAPE functions. Knowledge can be symptoms, policies, historical logs, metrics, topology information and etc. It is shared through registry, dictionary, database or etc. knowledge can be obtained through policies or be retrieved from an external knowledge source such as specific historical knowledge or can be either created or updated by autonomic manager itself. Monitor functions can create knowledge by collecting historical log files. Execution functions may update the shared knowledge to show the actions that were taken. Analyzing and planning functions may later use this information to enhance their decision-making process. Knowledge of application architecture in component-based systems can be used to detect foreign activities and to trigger counter measures [31]. Knowledge in a dynamic environment is not static. In a dynamic operational environment, shared knowledge among MAPE processes is subject to change. New techniques to obtain knowledge in a dynamic environment is discussed by Fisch et al. [50]. Their techniques benefit from communication of knowledge in a multi-agent system. By means of cooperation among intelligent agents, the agents are able to coop with the situations that they have not already faced.

• **Sensor:** Behavior and structure of the system need to be observed to be able to act on and modify the system. Basically sensors reflect the application behavior. Sensors in adaptive security aim at capturing changes that impact security. Methods such as packet filtering can be employed in observing necessary data. Sensors can be loggers that record system activities and events. The pattern of events that leads to an attack can be exploited in detecting the attack early on.

Furthermore capability of adding and removing sensors would let the system to plan on adding/removing sensors if necessary. For example when a particular behavior is observed, additional information might be needed before deciding the adaptation plan. In adaptive security, to distinguish a malicious attack from a suspicious

attack, supplementary information is demanded.

• **Effector:** A self-protecting system triggers the right countermeasure with or without minimal human interventions. Effectors trigger and execute the adaptation plan. A system can modify its own behavior or structure with the aid of effectors. Effectors can isolate a node, apply more strict rules in checking the packets that it sends, or even force to reboot or reinstall a node [31].

• **Managed resources:** The elements of a software system that are controlled with the adaptation manager are called managed resources. They are essentially what is found in software systems, for instance, CPU, database, server, etc.

It is worth mentioning that the main goal of this research is to realize the *Planning* process of the MAPE-K model. The other three processes in the MAPE-K (i.e., *Monitoring*, *Analyzing*, and *Executing*) are not the focus of this research.

2.2 Related Research Projects

Our work has taken shape in the context of a rich literature focused on (i) adaptive application security, (ii) modeling and analyzing security, (iii) decision making in SPS, (iv) application of game theory in cybersecurity, and lastly (v) attack-type uncertainty in SPS. Here, we relate our work to the most related projects in each of these fields.

2.2.1 Adaptive Application Security

Elkhodary and Whittle provide a survey on application adaptive security mechanisms [40]. The studied mechanisms are evaluated according to computational paradigm, reconfiguration scale, and conflict handling. The authors discuss decision making in application adaptive security from conflict resolution handling aspect; Whether the conflict is resolved by the user, the system, or both. Yuan et al. propose a taxonomy of SPS, the authors discuss adaptation decision making in terms of how adaptive decisions are made [133]. The authors also conducted a systematic survey on self-protecting software systems [132]. They categorized adaptation decision-making approaches into: human-driven, heuristic-driven, or algorithm-driven. An architecture-based self-protection approach is presented by Yuan et al. [135]. The authors describe several architectural adaptation patterns in order to detect and mitigate web application security threats. The described patterns are also realized based

on Rainbow [52] architecture-based adaptation framework. An adaptive rule-based malware detection at the host level is presented in [11]. The authors employ learning classifier systems, which is a combination of rule-based expert systems with evolutionary algorithms. Their approach is based on reinforcement learning in order to dynamically evolve decision rules. However, the focus of their work is not adaptivity at the application-level. A noticeable gap in the application-level adaptive security literature is the lack of systematic decision-making approaches.

SecuriTAS proposed by Pasquale et al. [99] enables software designers to model security goals and requirements of a system at the design-time. The model is used at runtime to analyze and plan for adaptation. Security concerns are modeled by Fuzzy Causal Network (FCN) from goal model, threat model, and asset model to represent the relationship among security goals, threats and assets. In SecuriTAS, decision making is supported by utility nodes in the defined FCN. Bailey et al. [7] propose a model-driven self-adaptive approach to access control. Their proposed run-time approach is based on model generation, transformation, and verification to provide assurances that the deployed adaptation conforms to the system requirements. Access control policies are associated with system resources, and access rights assigned to users; Access decisions are based on access control rules. An architecture-based self-protection is studied by Schmerl et al. [111] in which various denial of service mitigation strategies and their effect on quality attributes are evaluated using Rainbow self-adaptive framework [52]. The authors take advantage of decision theory and utility theory to provide a scientific basis for decision making in SPS. An automated DDoS attack mitigation platform called CAAP is presented in [9]. CAAMP, a software defined mitigation platform that dynamically mitigates the DDoS threats on applications on a public cloud using a private cloud. The above approaches, even though effective, rely on simple rules for action selection. Hence, they may fail to mitigate well-planned attacks.

2.2.2 Modeling and Analyzing Security in Self-Protecting Software

Modeling and reasoning with prioritization of Non-Functional Requirements (NFRs) to better making runtime decisions is of more interest in the recent years [3] [100] [101]. Modeling and analyzing security helps to collect and organize security related concepts such as security and non-security goals, threats to the systems, and attacks. Various modeling techniques have been proposed to model security aspects of software systems [119] [125]. These models consist of goal models, threat models, and

other similar models. Goal models capture both security and non-security goals of the adaptable system and the relation among goals of the system and goals of (malicious and regular) users.

An important issue in software security is analyzing the trade-offs among the competing goals of multiple actors (e.g., system admin, malicious user). In order to model goals of the software system and the attacker, there are well-studied goal models in the literature that can be employed (e.g., [12] [62] [80] [93] [97] [124]). Such modeling techniques provide proper basis for modeling and analyzing security trade-offs. However, dealing with runtime adaptation, a decision model is required to not only capture security analysis but also to accommodate payoffs and decision nodes.

A key challenge in software systems is continuous assurance of quality goals at runtime. Traditionally, satisfaction of quality goals is gained through a variety of requirement engineering and analysis at the design and the development time such as security requirement engineering [39]. Our proposed framework aims at providing an engineering approach to fuse adaptive security into software systems while considering quality goals of the software system as well as the malicious goals of an attacker.

2.2.3 Decision Making in Self-Protecting Software

One of the key challenges of feedback control loops in SPS systems is the action-selection process [77]. A quality-driven approach for enabling decision making in self-adaptive systems is discussed by Salehie and Tahvildari [108]. The authors propose an action-selection mechanism based on cooperative decision making. Various approaches for decision making in self-adaptive system have been proposed (e.g., [60]). A probability-based approach to support decision making in SAS systems based on dynamic decision networks is studied by Bencomo et al. [10]. The authors address the challenge of making adaptation decisions under uncertainty in SAS systems. Pandey et al. [98] present a novel hybrid planning approach for decision-making in self-adaptive systems. They deal with potentially conflicting requirements of timeliness and optimality of adaptation plans. Although there are many solutions for making adaptation decisions in SAS, decision making in SPS systems have not yet received the full attention they deserve.

The knowledge incorporated in selecting proper actions in these techniques is gained from the system itself and/or its environment. The destructive nature of

attacks distinguishes the decision making in SPS from the decision making in the other forms of SAS. SPS should be able to fuse the goal and strategies of attackers in its decision-making process so that it can take fast and valid reactions to sudden and unforeseen changes in strategies of attackers. Moreover, the goal of decision making in SAS is to select the adaptation action that helps to reach the system to its optimal state. However, in SPS the goal of the deciding process is to discover the adaptation action that directs the system to the safest state during runtime.

A variety of decision mechanisms in autonomic computing systems and more specifically self-optimizing systems is investigated in [86] and a comparison of the approaches for decision making is provided. Moreno et al. [92] present an approach for proactive latency-aware adaptation that makes fast adaptation decisions while producing similar results compare to another approach based on probabilistic model checking. A decision making strategy consists of a set of strategies (adaptation mechanism) and criteria (attributes presented in architecture layer) to achieve certain objectives (goals) [51]. Most approaches in SPS, implicitly fused these three dimensions as a set of rules or policies.

In general, decision making to provide cyber defense is an ongoing challenge. The work by Gonzalez et al. [56] aims at providing a better understanding of dynamics of cyber-attack and defense actions through simulations of cybersecurity scenarios using a multi-agent cognitive model framework. In the field of self-adaptation software systems, different techniques (from probabilistic models [10] to goal-driven models [110]) for making adaptation decisions are investigated. However, decision making in SPS systems has not yet received the full attention it deserves. A systematic survey of SPS systems [132] highlights the lack of sophisticated decision-making techniques in these systems. More recently, the approach presented by Barna et al. [8] uses a performance model to predict the impact of arriving requests; a decision engine adaptively generates rules for filtering traffic and sending suspicious traffic for further review. They use CAPTCHA puzzle test [126] to verify the legitimacy of users.

Bailey et al. in [7] propos an SPS system to protect the system against insider threats, by adapting the access control policies associated with system resources, and access rights assigned to users. Both [8] and [7] assume that the SPS system is aware of the type of attacks at the time of adaptation decisions. Data mining techniques have been extensively applied to provide security. Most research works have focused on intrusion detection at network and host levels (e.g., [76]) and malware detection at source code and executable levels (e.g., [113]). Recently, detecting malicious behavior at the architectural-level has been investigated. In [131], Yuan et al. took advantage of automatic mining of software component interactions to identify

potential malicious (abnormal) behavior in SPS systems. While developing threat detection approaches (based on mining data collected from network traffic, source code, or software component interactions) is an active research area, decision making in SPS systems is performed using traditional approaches.

Despite the central role of the decision maker in mitigating application-layer attacks, the impact of *attacker strategy* has not been directly modeled in the decision-making process of SPS systems. Consequently, it is not incorporated to drive the action-selection process.

2.2.4 Application of Game Theory in Cybersecurity

A variety of mathematical theories can be used to model and analyze cybersecurity [36]. In dynamic systems, control theory is beneficial in formulating the dynamic behaviour of the systems. In contrast to these approaches, game theory provides rich mathematical tools and techniques to express security problems. Roy et al. [104] emphasize that game theory can provide us with a mathematical framework for analyzing and modeling network-security problems. The applicability of game theory relates to the essence of game theory that deals with problems where multiple players with contradictory objectives compete with each other. Game theory continues to get more attention in different disciplines including IT security. Security games are employed as a basis for decision making as well as predicting the behavior of attackers [4].

Definitions of some basic game theory terms (refer to [4] [35] [83]) are presented in order to help readers better understand game theory. Security games consist of multiple *players*, a *set of actions* that can be taken by each player, and *payoffs* of taking any of these actions. The solution to these games is known as *Nash equilibrium* [96] which is a satisfactory solution concept that non of the players can gain further by changing their strategies. In multiple-person decision-making scenarios, the players are cooperative or non-cooperative competing for limited resources. Security games can be modeled as *non-cooperative games*, in which, players make decisions independently. A *static game* is a one-shot game where no player is allowed to change the strategy. Wu et al. design a static game for bandwidth depletion attack for DoS and DDoS [128] attacks. The existence of Nash equilibrium in the modeled game represents the best strategy for both players. In *dynamic games* each player can change their strategy during the game.

Game-theoretic models formalize the decision making in multiple players problems. Hence, concepts of game theory can be leveraged in the decision-making engine

of SPS systems. Security games allow players (the defender and the attacker) to develop a systematic strategy based on formalized methods. In security games, players do not have access to each others payoffs; therefore, they observe the opponent's behaviour and estimate the result of their action. Game-theoretic models are extensively discussed in the *network security* literature [58] [81] [88] [104] and more recently in the Internet of Things literature [75]. The applications of game-theory in *software security* has become more active in the recent years.

Game theory provides a quantitative approach in which the knowledge of players is expressed by mathematical models. Haley et al. [59] distinguish cybersecurity as a non-zero-sum game. In zero-sum games, your gain is your opponent's loss [35]. However, in cybersecurity, this is not necessarily true. An attacker may just be experimenting with a scenario for excitement and have no serious intention, even though the attack may cause loss of revenue for the attacked system. Recent efforts have attempted to address the challenges of understanding human factors in cyber defense. Gonzalez et al. [56] introduce a framework called CyberWar Game which employs game theory and simulates cybersecurity scenarios using a multi-agent cognitive model. The authors draw insights from studying societies in which agents exhibit human boundedly rational characteristics. They focus on addressing dynamics of decision-making in cybersecurity at a very high-level and is not applicable to software security.

The authors in the book chapter [15] show how Stochastic Multiplayer Games (SMG) can be used to mitigate different types of uncertainty in contexts such as self-protecting systems, proactive latency-aware adaptation, and human-in-the-loop. SMG has been employed to enable developers to approximate the behavioral envelope of a self-adaptive system by analyzing best- and worst-case scenarios of alternative designs for self-adaptation mechanisms. Similar to the SMG analysis in this paper, the interplay between a self-adaptive system and a potentially adversarial environment as an SMG, and the analysis that accounts for the strategy of the adversary when selecting counter-measures has been described in [14] [16] [17] [18] [111] [112]. Yet, the framework and SMG analysis in this thesis is different from SMG analysis in those studies by incorporating intention and strategies of attackers with the action-selection process.

2.2.5 Attack-Type Uncertainty in Self-Protecting Software

Uncertainty can be defined as the difference between information that is represented in an executing system and the information that is both measurable and available at a

certain point in the system's life-time [54]. A definition and taxonomy of uncertainty within the context of a dynamically adaptive systems is presented by Ramirez et al. [102]. Furthermore, a summary of existing techniques and insights into addressing uncertainty in self-adaptive systems is provided by Cámara et al. [13]. They present a method to represent different types of uncertainty in MAPE-K self-adaptive systems. Their analysis results show that although uncertainty-aware adaptation does not guarantee to perform better than non-uncertainty-aware adaptation in all cases, but rather in most of the cases.

Uncertainty in security games could be about information that the defender has with regard to the attacker. In [123], authors investigated how human defenders behave under several levels of uncertainty and various types of attack strategies. They conclude that defense algorithms would be more efficient if they are adaptive to the attacker actions. More recently sensing uncertainty in decision making for self-adaptation is studied by Cámara et al. [19]. Their analysis is based on model checking of stochastic multi-player games. Their results show that although uncertainty-aware adaptation is not guaranteed to perform better than uncertainty-ignorant adaptation in all cases, it does perform at least comparably in all cases.

Incorporating attack-type uncertainty into the decision-making process is studied in the field of network security, e.g., wireless ad hoc networks [85], Mobile Ad hoc Networks (MANETs) [79], and network protection [53]. Capturing intentions of attackers and the alternative ways they can be realized through an attack is studied in the field of requirement engineering. Alexander [2] advocates using misuse and use cases together to conduct threat analysis during requirements analysis. Van Lamsweerde [124] provides constructive guidance in early elaboration of security concerns. In his paper, a requirement engineering method for building an intentional anti-model is provided. The anti-model produces vulnerabilities and capabilities required for achieving the anti-goals. Although security requirement engineering provides models that elicit attackers' goals and intentions at the requirement engineering stage, runtime action selection in an SPS system requires the support of runtime models.

Only a few cybersecurity approaches consider adversary-type uncertainty in their decision models. For instance, a recent work by Garnaev et al. [53] studied the incorporation of adversary-type uncertainty in network security protection. They discuss the dilemma a network defender faces due to limited resources: either to focus on increasing defense of the most valuable nodes or to defend all the nodes while reducing the level of defense of the most valuable ones. They suggested a Bayesian game in which the probability of the adversary's type can be considered

as a sub-scale in the scale of threat levels. While the authors in [53] address the uncertainty in network security, their modeled matrix game is specific to a network of nodes in a communication network and cannot be applied in software security.

Another work, by Li and Wu [79], describes a dynamic Bayesian game between regular and malicious nodes in Mobile Ad hoc Networks (MANETs). In their modeled game, the regular node forms beliefs and measures uncertainty to evaluate the type of opponent. The game is studied to choose the probability of cooperating with the opponent or to report the malicious node. Additionally, a malicious node evaluates the risk of being caught and uses its flee strategy to avoid punishment. The game modeled by Li and Wu [79] is specific to MANETs and cannot be applied in the field of software security. In wireless ad hoc networks, the uncertainty of a defender about his type of opponent (regular or malicious) is formulated in both static and dynamic Bayesian game contexts [85]. The modeled game is specific to IDS implementation in the network layer and cannot be employed in the field of software security, and more specifically, for addressing security goals at the application level. Liu et al. [84] developed a game-theoretic formalization that can capture the inherent interdependency between an adversary's intent, objectives, and strategies and a defender's objectives and strategies in such a way that adversarial objectives and strategies can be automatically inferred. However, in their game model the security goal preferences of the system are not considered in the decision model.

Recent literature in security games incorporates payoff uncertainty [94] and adversarial uncertainty [95] using Stackelberg security games. Their modeled security game is general and not specific to cyber security. Fielder et. al. [49] address the challenge of making better security decisions by the aid of a game-theoretic model that captures essential characteristics of resource allocation decision making (i.e. system administrators' time) to prevent data loss and defend system and network assets of an organization.

As discussed above, the state-of-the-art either focuses on incorporating the uncertainty about adversary types in (i) specific network security scenarios or (ii) general security games. However, the focus of this article is on software security and fusing the uncertainty about adversary types into the decision-making model of an SPS system. More specifically, the benefits and costs of protecting/mitigating security goals in a software system are incorporated in the proposed decision model.

2.3 Summary

This chapter gives a comprehensive review of principles for this research. We have extensively reviewed the state-of-the-art research in SPS systems and raised relative research challenges and gaps in this area. A revealing insight from this overview of related research projects is the absence of decision-making techniques that capture all the possible knowledge from the software system and its operational environment (including its users) and incorporates such knowledge in making effective adaptive decisions.

Observations from the state-of-the-art research can be summarized as follows: (i) the developed underlying decision making in the current literature is based on utility values that are defined by the use of policies or goals, (ii) machine learning techniques mostly concentrate on the detection of attack followed by simple rule-based attack mitigation, and (iii) most state-of-the-art approaches assume that the attacker initiates and continues the attack with the same strategy.

It is worth mentioning that the problem of *"How to design a dynamic decision making approach that incorporates the strategy of the attacker"* has not been explored neither in academia not in industry. One solution is exploiting game theoretic approaches. In spite of that, the application of game theory as a potential solution for decision making in SPS systems has not been investigated. This PhD research employes game-theoretic techniques in the decision making of SPS systems. In the next chapter, we present our approach to address incorporating the strategy of attackers in the decision-making of SPS systems.

Chapter 3

A Framework for Decision Making in Self-Protecting Software Systems

Securing software systems is a critical concern as these systems are involved in almost every aspects of today's lifestyle. Moreover, the growing number of dynamic and well-planned attacks make it more challenging to reach the desired level of security especially in complex and interconnected software systems. Traditional software security techniques focus on threat detection approaches and building Intrusion Detection Systems (IDSs). However, dynamic security attacks call for fast reacting software systems that not only detect, but also mitigate threats. Building *Self-Protecting software* (SPS) systems is a response to these demands. The main objective of SPS systems is to satisfy software's security goals and requirements by triggering appropriate countermeasures. In this research, we provide a framework to facilitate decision-making in SPS systems in such a way that the triggered countermeasures are aligned with the security goals of the software system.

To devise a game-theoretic decision-making framework, we start by presenting a motivating scenario 3.1. Followed by defining the three-phases of the proposed framework in Nutshell in Section 3.2. Then we elaborate on each phase of the framework in Sections 3.3, 3.4, and 3.5.

3.1 A Motivating Scenario

Two types of attacks that are major threats to a software system and significantly difficult to detect and mitigate are: (i) slow application-aware DoS attacks [8], and (ii) insider attacks [21]. These attacks differ in the essence of their damage to the software systems. The former targets the *availability* of the software system and the latter threatens its *confidentiality*. Interestingly, both application-layer DoS and insider attacks share a similar scenario to fulfill their malicious goals; a user or multiple users keep downloading documents to either (i) overload the server, or (ii) gain access to confidential information in the system. In general, the scope of the attacks that we aim at addressing in this thesis is limited to the attacks that share similar scenarios and as a result confuse the IDS in determining the actual type of the attack. DDoS attacks can be modeled as the multiple DoS attacks occurring at the same time and each DoS attack is treated separately.

As a motivating example, consider a software system that provides mitigation against both application-layer DoS and insider attacks by issuing a CAPTCHA test [126] against DoS attacks or removing the access permissions of insider attackers [7]. In this system, suspicious number of downloads is detected, but the attack type (an application-layer DoS attack or an insider attack) is not yet distinguishable. Consequently, even though the software system is capable of mitigating both types of attacks, the decision-making engine fails to choose the proper countermeasure as soon as the malicious scenario is detected. Hence, before the attack strategy is detected, the system may (i) *take a random* countermeasure, or (ii) *take a fixed* countermeasure. For this reason, the SPS system may either take a wrong response (which does not address the attack properly) or delay the selection of the countermeasure until the attack type is clear to the SPS system. For example if an application-layer DoS attack continues, the system's availability noticeably drops, or if an insider attack is in progress, unauthorized access of sensitive information will be noted eventually. This delay works very much to the advantage of the attacker and its malicious goals.

What differentiates these two types of attacks from each other is the *intention* behind them. In this motivating example, even though both application-layer DoS and insider attacks exhibit a similar malicious behavior to the monitoring infrastructure, the attackers intentions are entirely different. When initiating an application-layer DoS attack, the objective of the attacker is to break down the system and make it unavailable to its normal users. In the event of an insider attack, the goal of the attacker is to gain access to sensitive information in the system. Accordingly, defense strategies for these two attack types substantially differ from each other.

This fact highlights the importance of the *correlation among defense strategies* in the underlying decision-making engine in the case of *attack-type uncertainty*.

In our framework, we aim at providing this correlation by incorporating the intentions and strategies of attackers into the decision model. The intentions of attackers can be modeled using soft-goal models such as Soft-goal Interdependency Graph (SIG) [29]. Moreover, the correlation among attack strategies and defense strategies is captured by modeling a two-player game and the impact of strategies are instrumented with utility functions.

3.2 The Game-Theoretic Decision-Making Framework in Nutshell

Our research objective is to provide decision making at runtime to adapt the system in supporting security goals while incorporating the likelihood of an attack. To achieve our research objective, we need to resolve several issues. First, we need to design systematic models to represent, capture and analyze the impact of countermeasures on security and non-security quality goals as well as on possible goals of attackers. Second, given a list of countermeasures that can be taken by the adaptable software and possible attacks that threat the adaptable software, we need a decision-making mechanism to reason which countermeasure is suitable to take considering strategy of the attacker. Third, we should be able to assess the adaptation plan to evaluate the success or failure of the adapted countermeasure. These issues can be accomplished by addressing the following set of research questions. We need a methodology that provides notations, techniques and guidelines in developing an SPS system. Moreover it should also allow for traceability and evaluation of security goals during the runtime.

Fig. 3.1 depicts the phases of our decision-making framework. The framework aims at exhibiting a plug-and-play capability to adapt a game-theoretic technique that is the best fit considering the security goals and requirements of the software system. The phases are summarized as:

• **Modeling: Goal-Oriented Model**. In SPS systems, the knowledge incorporated in the decision models is not limited to the software itself and their running environment. The users of a system are the potential attackers to a system. Hence, the knowledge that captures the behavior (and ultimately trustworthiness) of the users can enhance the quality of the decision making. In this research work, we model the

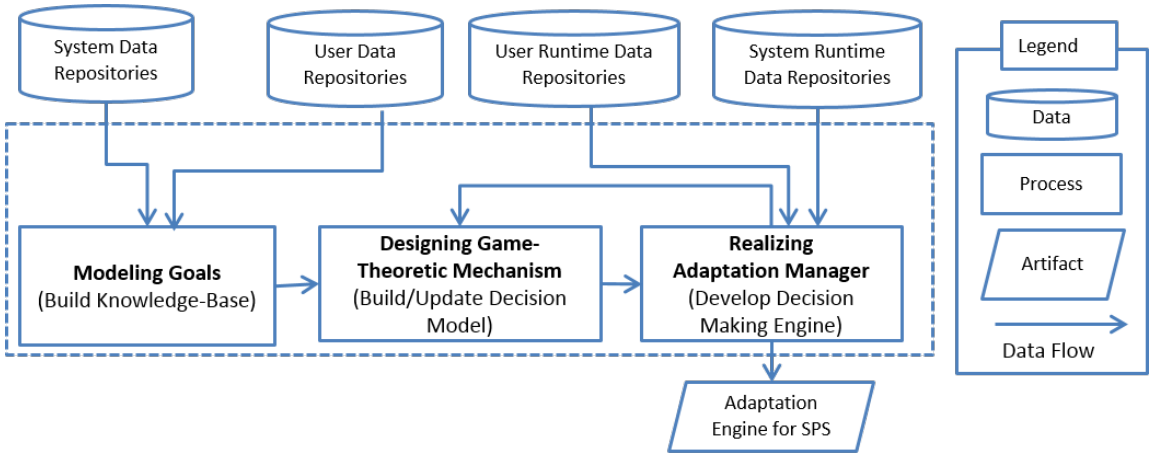


Figure 3.1: Game-Theoretic Decision-Making Framework

required knowledge by exploiting goal-oriented modeling techniques. These techniques facilitate trade-off analysis among goals and provide a suitable base for dealing with security. Goal-oriented models captures the required knowledge not only from the adaptable system's perspective but also from malicious user's perspective. The analysis of the trade-offs among goals in these models will further facilitate the analysis of security decisions for the decision-making engine in a SPS.

• **Designing: Game-Theoretic Decision-Making Mechanism**. The second phase of the research is designing the planning process of the SPS. Decision making in competitive scenarios can be modelled as a game between the two players. Game-theory is a natural fit to quantitatively analyze multi-players actions and utilities. In designing the decision-making engine, we leverage game-theoretic algorithms. They are distinguishable from other algorithms in considering the strategy of the opponent when choosing the next action. Hence, game theory aids in trade-off analysis of security countermeasures and possible attacks.

• **Realizing: Adaptation Manager for SPS**. The third phase of the research is the development process in order to realize the first two phases. The first step of this phase covers collecting the required information to build the goal-oriented model and mapping the model to a game-theoretic technique. Next step is to develop the decision-making mechanism. The last step in this phase is to evaluate and test the planning process. Evaluation needs to be performed in a complete adaptation engine interacting with other adaptation processes.

The research objective is to provide action selection at runtime to adapt the

system in supporting security goals. The plug-and-play feature in our framework is the capability of the framework to employ (i) different *goal models* (in the first phase) depending on the properties of the available data that can be used in the model and (ii) various *game-theoretic techniques* (in the second phase) based on the security goal and requirements of the software system. The framework aims at: (i) capturing trade-offs among security and non-security goals, (ii) quantifying the impact of an attack when selecting a countermeasure, and (iii) choosing a plan of actions incorporating likelihood of an attack by employing a game-theoretic decision-making technique. The following sections give more details on each phase of the framework.

3.3 Modeling: Goal-Oriented Model

Security issues arise when a user of the system strives to achieve a goal that intentionally or unintentionally threats the quality goals of the software system. In our research approach, we consider the adaptation manager as a player in a game that aims at protecting assets of the system. Besides, a malicious user of the system is considered as a player that competes against the adaptation manager to exploit system's assets with the intention of breaking down the system. Therefore, in the first phase of our research, the goal is to develop an goal-oriented model that captures the relation between various concepts of both players. The model should be able to represent goals and actions of the software system and specially their effect on goals and actions of malicious users. Relations such as dependency or conflict among concepts are recognized at this stage.

The first issue, when building a knowledge-base decision-making framework, is representing the required knowledge. The knowledge includes goals, attributes, states, and strategies. The objective of developing a goal-oriented model is to predict the intentions and strategies of attackers based on available information. In the following subsection, we explain in more details the input concepts and their relations in developing a goal-oriented model for SPS.

3.3.1 Concepts and Their Relations in a Goal-Oriented Model

In this section, we elaborate further on the concepts comprising the goal-oriented model.

(a) **Quality Goals and Malicious Goals**: Incorporating goals in the adaptation model has the advantage of being *traceable* and *trustable* [110]. It facilitates analyzing and tracing goals at runtime and provides transparency of goals and requirements. Specifying the goals of the players (adaptation manager and malicious user) helps to answer the following question: "What are criteria which players bases their decision upon?". These criteria can be measured during the runtime of the system. Therefore, the decision-making process can quantify goals at runtime and consider their level of satisfaction when making an adaptive decision.

(b) **Countermeasures and Attacks**: Depending on the vulnerabilities of the software system, various attacks threaten the system and ultimately diminish satisfaction of quality goals. A set of countermeasures choose to be developed in the system aiming to prevent or mitigate such threats. This set of countermeasures may include weak adaptation actions, such as parameter adjusting, or strong adaptation actions, such as adding or removing system components [109].

(c) **Adaptable Software State and User Category**: Both software states and user categories can be represented in terms of various attributes. Values of these attributes can be discrete or continuous. Discrete attributes, such as the load of the system, can take certain values (e.g. "high", "normal", and "low"). Continuous attributes indicate a measurement at a certain point. For example, the number of requests of a user per hour.

Determining the maliciousness of a user is a challenge by its own and is out of the scope of this research. We can use simplistic behavior monitoring to determine the maliciousness of the action taken by a user. For instance, measures to determine maliciousness of a user can be the amount of traffic generated by the user or the sensitivity of the data that has been accessed by the user. Other approaches can be employed to detect suspicious activities. For example, Lamba et al. [74] describe a model-based approach to cluster sequences of user behaviors within a system and to find suspicious, or anomalous, sequences. Yuan and Maled [134] detect anomalous behavior by mining software component interactions from system execution history. Zawawy et al. [136] discuss a root cause analysis technique that can be used to identify the causes of failures utilizing a probabilistic reasoning technique that is based on the use of Markov Logic Networks.

(d) **System Utility Function and User Utility Function**: System and user

utility functions are essential to enable quantitative decision-making. The information in utility functions aim at: (i) quantifying cost and benefits of *countermeasures* and *attacks*, and (ii) specifying preferences of each *quality goal* and priorities of *malicious goals* while considering the *state* of the adaptable software and the *category* of the malicious user who is sending suspicious requests to the system.

The relations among the above mentioned concepts need to be considered in the goal-oriented model that will be developed, namely: (i) relations between denial and satisfaction of software's quality goals and goals of the malicious user, (ii) relations between success and failure of attacks and countermeasures, and (iii) relations between software states and software quality goals and between user categories and malicious goals. More details are given in the followings:

(**a**) **Relation Between Quality Goals and Malicious Goals**: Goals determine the criteria of selecting actions at runtime. Actions that satisfy higher priority goals are preferred over other actions. The two involved players (adaptation manager and malicious user) have conflicting goals. Hence, specifying these conflicts and the level of their impact on each other help the adaptation manager in making a finer decision, considering various attack scenarios that a malicious user may take.

(**b**) **Relation Between Countermeasures and Attacks**: Different software systems have different vulnerabilities that depend on their design and their development. After identifying vulnerabilities of the system, the next step is to determine potential attacks that can exploit them. Possible countermeasures of the system and potential attacks can be related through vulnerabilities of the system. Thus, by realizing the vulnerabilities of the system, we can define a set of countermeasures that system can adapt to protect such vulnerabilities or eliminate them. At the same time, malicious users may take advantage of such vulnerabilities and target their attacks to harm the system by exploiting one of these vulnerabilities. Identifying types of attacks that a malicious user can trigger is beneficial in preventing them from happening.

Capturing the relations between countermeasures and attacks allow security decisions to be made rationally.

(**c**) **Relations Between Software State and Quality Goals and Relations Between User Category and Malicious Goals**: To connect the running

system and its users to the adaptation manager, the state of the running system as well as its current users' behavior need to be represented to the adaptation manager. We aim at capturing the relations between software states and quality goals along with the relations between user category and malicious goals. The preferences of quality goals may vary depending on the state of the software. For example, at certain states, high availability goal is the essential security goal but in other states, high performance goal has higher priority.

Depending on the categories of users (such as insider attackers or careless users), their intention and hence their malicious goal varies. Characterizing users helps to identify their goals and consequently capturing the likelihood of the actions that threaten the system at any point of time. We plan to relate: (i) the states of the system to its quality goals, and (ii) the categories of users to their possible malicious goals.

In SPS systems, unlike other self-adaptive software systems (self-optimization, self-configuration, or self-healing systems), the adaptive decision is highly dependent on the strategy of the attacker. Hence, a goal-oriented model that captures attacker's strategies is required. In general, two critical points are required in the potential goal-oriented model that will be used: (i) capability of modeling more than one agent's decision situation, and (ii) capability of capturing the relationship between quality goals and decisions.

The purpose of employing a goal-oriented model is to provide the adaptation manager with the knowledge required in making a systematic decision. This phases of the framework is carried out offline to build the goal model. However, if required, the model can be updated during runtime of the software system as the security goals and requirements change. In the next section, we describe the second phase of the research which relies on the developed goal-oriented model in the first phase.

3.4 Designing: Game-Theoretic Decision Making Mechanism

Envisioning all possible strategies that can be taken by a malicious user is not possible at the design-time. This challenge is addressed by utilizing game-theory in decision making of SPS systems. The purpose of this phase is to design a game-theoretic mechanism using the goal-oriented model produced in the previous step. The model

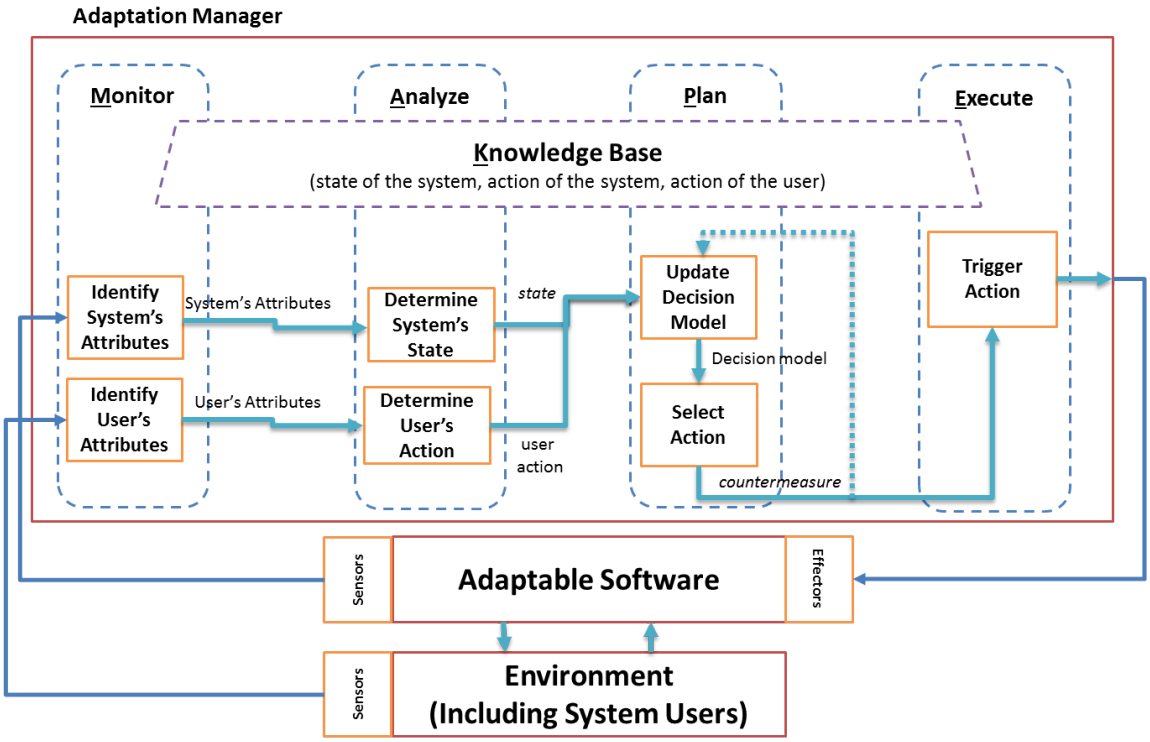


Figure 3.2: Retrofitting MAPE-K Model to Incorporate Game-Theoretic Design

guides in designing the decision model. The decision model is constructed using game-theoretic algorithms. This phases of the framework is carried out offline to design the decision-making algorithm and implement it. However, the runtime data will be used during runtime as the inputs of the decision-making algorithm to select a countermeasure and update the decision model.

In SPS systems, the adaptation mechanism consists of five main components, based on MAPE-K reference model, presented by IBM in [71]: *Monitoring*, *Analyzing*, *Planning*, *Executing*, and *Knowledge-Base*. In our approach, we incorporate modeling the knowledge about the intentions and behavior of users into the MAPE-K loop as part of the knowledge-base. The retrofitted MAPE-K reference model is illustrated in Figure 3.2.

In this retrofitted model, the monitored data is originated from both the adaptable software and its environment which encompasses system users. The raw data is then filtered or transformed (if needed) to identify information of the system and users in the forms of various attributes. Subsequently, the attributes are passed

to the analyzing process. It encapsulates two main aspects of the decision-making process: (i) the states of the system, and (ii) the actions of the user. The third element of the decision-making process is the learning factor which is the effect of the previous action that was executed.

One of the important aspects of a decision-making mechanism is the capability of learning. It guides determining the benefit/loss after employing the adaptation action that is selected by the system in respond to the malicious user's action. It requires calculating the effect of the last action taken by SPS. This effect can be characterized based on the performance of the system or the maliciousness degree of the user:

- **Learning from performance of the system**: One way to find the benefit of the adopted action is by measuring the performance of the system. For example, measuring the amount of increase or decrease in the load of the system can be an estimation of the effectiveness of an action and hence, the reward value.
- **Learning from maliciousness degree of the user**: Another option to capture the reward value can be based on the user's request and/or the user's history of requests. For example, the reward of dropping skeptical requests are higher than the reward of dropping normal requests to the system.

In dynamic learning of malicious users' intention and strategy, incorporating the degradation or enhancement of the performance of the software system can be effective. It is also worth mentioning that the performance of the system can change due to reasons other than the system being under attack. Hence, in some types of attacks, such measurement is not a good indicator to detect an attack. In such situations, characteristics of requests issued by users are better candidates to evaluate the feedback from taking an adaptation action.

The feedback from the previous adaptation action can be either by quantifying the success (positive feedback) or failure (negative feedback) of the action. The result of learning from positive or negative feedback is not necessarily the same. In some systems, learning from success of the adaptation action is more effective than learning from its failure or vice versa.

3.5 Realizing: Adaptation Manager for Self-Protecting Software Systems

The objective of this phase is to realize the first two phases in building the adaptation manager for SPS and furthermore implement and evaluate them. The steps in this phase of research are summarized below:

(**a**) **Building Goal-Oriented Model**: The first step is responsible for collecting the required information and provide a structure for this set of information. The modeling process gathers the required information from two repositories of *security policies* and *vulnerability assessment*. The developed model aims at capturing the relevance of various input information. It incorporates priorities and preferences of quality goals and their related actions. Such information can be gained from security policies repository. This process analyzes the security and non-security goals and requirements of the system and the impact of adaptation actions on those goals. In addition to modeling the system information, the advantage of goal-oriented model is that it also recognizes possible malicious goals and attacks for each vulnerability that exist in the system. This information can be provided from vulnerability assessment repository. Since, the essence of the to-be-developed adaptation model is capturing more than one agent's goals and actions, the resulted model is able to represent the impact of defense strategies on the attacker's strategy. The resulted goal-oriented model is the input to the next process.

(**b**) **Developing Game-Theoretic Decision Making Algorithm**: Having a decision model that captures the deciding criteria, the next step is to incorporate it into the decision making mechanism that is in charge of finding the sequence of actions that satisfy security goals. The decision making strategy is designed and developed based on the developed decision model.

(**c**) **Implementation**: After building the model and developing the decision making algorithm, the decision making mechanism need to be implemented as part of adaptation manager planning process. The adaptation manager can either be implemented internal or external to the adaptable software.

(**d**) **Evaluation**: Part of evaluating the action selection algorithm is evaluating the satisfaction of security goals and requirements that are specified in the security policies. The evaluation is possible by a feasibility study on various types of

attacks at runtime and by analyzing the impact of selected countermeasures on utility function of the system and malicious users.

The third phase of the framework is carried out partly offline (building the goal model, designing the decision-making algorithm, and implementation) and partly online (evaluation). In general, the three phases of the framework are performed throughout the Software Development Life Cycle (SDLC) [90]. From analyzing the security and non-security goals, to designing the game-theoretic decision model and implementing the decision-making engine and evaluate it.

The framework is scalable in terms of both the goal model and the decision model. The quality goals that can be added to the goal model are limited. Hence the goal model is scalable to capture as many quality goals are required. The decision model is also scalable as the number of defense and attack strategies are not overwhelmingly large. The scalability of the realization phase can be influenced by the number of the malicious users. The framework monitors and analyzed each malicious user and selects a countermeasure per user.

The complexity that is introduced to the software system by the above decision-making framework is mainly related to the complexity of the designed game-theoretic technique in the second phase of the framework. In general, the complexity of computing two-player Nash equilibria is studied in detail by Chen et al. [26] and Daskalakis et al. [34]. The modeled two-player games in our framework have limited number of actions for each player and hence computing the Nash equilibria is straightforward.

3.6 Summary

This chapter introduced the high-level framework for decision-making in SPS systems. The framework aims at providing a systematic approach in engineering an adaptation manager for SPS systems. The approach aims at exhibiting a *plug-and-play* capability by it ability to adapt a decision-making technique that suites security goals and requirements of the software. It enables employing decision-making techniques capable of learning malicious user's strategy and adapting accordingly. Moreover, it considers various strategies that can be taken by the involved players (adaptation manager and malicious users of the software system) before making the adaptation decision. In the following three chapters, we investigate three different game-theoretic techniques that are employed with the aid of the proposed plug-and-play decision-making framework.

Chapter 4

Decision Making using Stochastic Games

Software systems play an increasingly important role in critical infrastructure, government and everyday life. These systems are being designed and deployed, however, security often is left for later. To make matters worse, the attacks that threat such systems are becoming more sophisticated and are becoming harder to detect and mitigate. Despite such challenge, today's state-of-the-art in software security consists of solutions that address a particular security attack that targets Confidentiality, Integrity, or Availability (CIA) security goals. Mostly, these solutions satisfy one or two of these goals [132]. However, in the real world, each attack has a different impact on security (and non-security) quality goals of the software system based on the vulnerability being exploited by the attacker. Acknowledging this impact on quality goals is beneficial to choose the proper defense strategy. Interestingly, various types of attacks (even though distinct in their nature) can share similar attack scenarios (behavioral patterns). In such cases, the targeted software system fails to select the appropriate strategy when facing an attack scenario that is common among different attack strategies. Consequently, a proper countermeasure cannot be applied until the nature of the attack is clear to the software system. In other words, even though a software system has the capability to protect itself against each individual attack, it may be incapable of acting when faced with attack type uncertainty. Such uncertainty is more probable when the attack is in its early stages and the Intrusion Detection System (IDS) lacks enough symptoms to identify the type of the attack.

Software systems are a target for different attack strategies such as Denial of Service (DoS) attacks [91], and insider attacks [21]. A Self-Protecting Software (SPS)

system is capable of detecting security threats and mitigating them through runtime adaptation techniques [135]. Mature and effective defense mechanisms are proposed to secure a software system [132] [40] [48]. There are extensive research studies on detecting and mitigating each type of attacks individually [126][111][8][7]. However, attack strategies can exhibit similar scenarios which results in uncertainty about the type of the attack. There is little work on mitigation strategies that intervene in a coordinated manner [114]. The lack of integrated defense strategies gives rise to triggering countermeasures that do not properly mitigate when facing with attack-type uncertainty. Although the continuation of an attack can clarify the strategy of the attacker eventually (when enough data is gathered to analyze the attack strategy), the delay between attack detection and mitigation can be easily exploited by attackers to significantly damage the software system. However, state-of-the-art solutions suffer from not systematically incorporating the strategies of the attackers in their decision-making model.

Most research works that consider the strategies of attackers have focused on network-layer security. Incentive-based modeling and inference of attacker's intent, objectives, and strategies and a game-theoretic formalization is presented by Liu et al. [84]. However, their modeling is specific to the security at the network level. Recent work proposed by Zonouz et al. [137] considers a game-theoretic intrusion response engine by modeling a Stackelberg stochastic game. Nonetheless, their proposed response engine targets security at the network level and is not applicable to security at the application level.

In [45], we argued that the application of game-theory in Self-Protecting Software (SPS) systems demonstrates promising results due to the incorporation of attack strategies in the decision model. In this chapter, we introduce the idea of an *Incentive-Based Self-Protection (IBSP)* framework that exploits game-theory to address the aforementioned challenges. The IBSP framework extends the proposed framework in Chapter 3. IBSP follows the three high-level phases as discussed in Chapter 3. In IBSP, by incorporating malicious intentions of attackers into the decision model, we aim at selecting a countermeasure that is aligned with the quality goals of the SPS system while minimizing the satisfaction of the malicious goals of attackers.

Our objective is to model and design a decision-making engine that reasons about: (i) the impact of these actions on quality goals of the software system, and (ii) the impact of the available countermeasures on malicious goals of attackers. The novelty of our contribution is the incorporation of the *malicious goals and intentions of attackers* into the decision-making process of SPS systems.

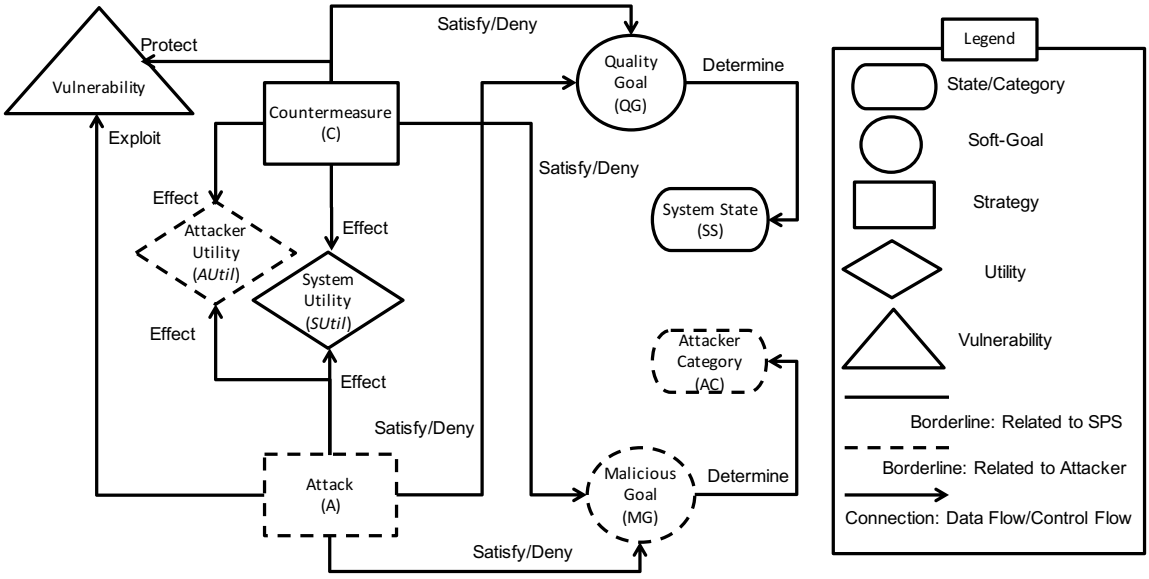


Figure 4.1: Required Concepts in our Incentive-Based Self-Protection Framework

The rest of this chapter is organized as follows. Section 4.1 presents the concepts and notations used throughout this chapter. Section 4.2 gives an overview of the extended framework employing the stochastic game technique. Section 4.3, 4.4, and 4.5 dive deeper into each phase of the extended framework: Modeling, Designing, and Realizing phases respectively. Section 4.6 demonstrates the obtained results of the evaluation. Finally, Section 4.7 summarizes the chapter.

4.1 Concepts and Notations

Table 4.1 shows the notations used throughout this chapter. Sets are shown by calligraphic letters such as $\mathcal{X} = \{X_i\}$. Hence, italic letters with a subscript (X_i) represent a member of a set. Function names are illustrated by italic letters e.g., $Func(list\ of\ inputs) \rightarrow output$. The utility values are shown by italic letters with a subscript representing the name of the approach. The percentage of the difference between two utility values is shown by italic letters that start with Δ and has a subscript for its name.

The main concepts involved in IBSP framework are shown in Figure 4.1. The SPS system and the attacker are defined by the same set of concepts while information related to the SPS system are illustrated with solid lines and information related to

Table 4.1: Summary of Notations Used

Notation	**Definition**
$\mathcal{A} = \{A_i\}$	Set of Attacks
$\mathcal{C} = \{C_i\}$	Set of Countermeasures
$\mathcal{AC} = \{AC_i\}$	Set of Attacker Categories
$\mathcal{SS} = \{SS_i\}$	Set of System States
$\mathcal{QG} = \{QG_i\}$	Set of Quality Goals
$\mathcal{MG} = \{MG_i\}$	Set of Malicious Goals
$\mathcal{G} = \mathcal{QG} \cup \mathcal{MG} = \{G_i\}$	Set of all Quality and Malicious Soft-Goals
QG_P	(Soft-Goal) High Performance
QG_U	(Soft-Goal) High Usability
QG_R	(Soft-Goal) High Reliability
QG_S	(Soft-Goal) High Security
MG_A	(Soft-Goal) Access to Sensitive Data
MG_B	(Soft-Goal) Breaking Down the Server
$\mathcal{SI} = \{SI_i\}$	A Set of Strategy Interdependencies
$\mathcal{SIG} = \{SIG_i\}$	A Set of Soft-Goal Interdependency Graphs for both Quality and Malicious Soft-Goals ($\mathcal{QG}$ and $\mathcal{MG}$)
$\mathcal{IMP} = \{++, +, ?, -, --\}$	A Set of Impacts $\{++ : Make, + : Help, ? : Unknown, - : Hurt, -- : Break\}$
$\mathcal{LAB} = \{\times, W^-, U, \natural, W^+, \surd\}$	A Set of Labels $\{\times : Denied, W^- : Weakly\ Denied, U : Unknown,$ $\natural : Conflict, W^+ : Weakly\ Satisfied, \surd : Satisfied\}$
$Label(SI_i, G_j) \rightarrow \mathcal{LAB}$	Label of SI_i on Soft-Goal G_j
$\mathcal{SLAB} = \{Label(SI_i, G_j)\}$	A Set of Labels for all SI_i in $\mathcal{SI}$ and all Soft-Goals G_j in $\mathcal{G}$
$Pref(G_i) \rightarrow [0..1]$	Preference of Soft-Goal G_i
$Sat(Label(SI_i, G_j)) \rightarrow [0..1]$ OR $Sat(\mathcal{LAB}) \rightarrow [0..1]$	Satisfaction of Soft-Goal G_j via Strategy Interdependency SI_i
$SUtil(SI_i) \rightarrow [0..1]$	Utility of Strategy Interdependency SI_i for the System
$AUtil(SI_i) \rightarrow [0..1]$	Utility of Strategy Interdependency SI_i for the Attacker
$AUS_{Name} \rightarrow [0..\infty]$	Accumulated Utility of the System via Mitigation Approach $Name$
$AUA_{Name} \rightarrow [0..\infty]$	Accumulated Utility of the Attacker via Mitigation Approach $Name$
ΔAUS_{Name}	The delta between two System Accumulated Utility Values
ΔAUA_{Name}	The delta between two Attacker Accumulated Utility Values

the attacker are represented with dash lines to be easily distinguished. Following concepts are considered:

• **Vulnerabilities**: By realizing system vulnerabilities, we can define a set of countermeasures that protect the system. Nevertheless, attackers may exploit those vulnerabilities and harm the system.

• **Countermeasures ($\mathcal{C}$) & Attacks ($\mathcal{A}$)**: The specific vulnerabilities of a software system lead to specific types of attacks and ultimately diminish satisfaction of its quality goals. Consequently, a set of specific countermeasures must be developed in an SPS to prevent or mitigate such threats. Countermeasures and attacks are paired to build Strategy Interdependencies (SIs). SIs capture the impact of the attacker's strategy while considering the system's selected countermeasure and vice versa.

• **Quality Goals ($\mathcal{QG}$) & Malicious Goals ($\mathcal{MG}$)**: We incorporate not only the *quality goals* of the system but also the *malicious goals* of the attacker. Incorporating quality goals of the system into the decision model has been studied in self-adaptive systems [10] as well as in SPS systems [59] [106] [99]. The Modeling of goals has the advantage of improving *traceability* and *trustability* [110]. In order to represent and reason about the malicious goals of attackers, they can be modeled with the aid of conceptual modeling approaches such as the extended version of the i* framework [130] proposed by Elahi et al. [38]. We assume that conceptual models of quality goals and malicious goals are provided from the security requirement engineering phase. This *design-time* information will be mapped to a *runtime* decision model.

• **System State ($\mathcal{SS}$) & Attacker Category ($\mathcal{AC}$)**: Combining different forms of evidence leads to more-precise attack detection. In SPS, various possible evidences are gathered from monitored attributes. We categorize these evidences based on the system runtime data and attacker historical data into two main groups: (i) *System State*: is determined based on the preferences of quality goals in the SPS system, and (ii) *Attacker Category*: is determined based on the preferences of malicious goals of the attackers.

• **System Utility ($SUtil$) & Attacker Utility ($AUtil$)**: Utility functions for the software system and attackers are essential to enable quantitative decision-making. We map the qualitative values, which represent the satisfaction of goals, to quantitative values between 0 (low) to 1 (high).

Our research objective is to design a framework that systematically models the impact of attack and defense strategies on quality goals and malicious goals. Such model can be mapped to utility values that guide the decision-making engine to

select a countermeasure in response to an uncertain attack type. In the next section we provide an overview of the proposed IBSP framework.

4.2 A Stochastic Game Approach

An SPS system is capable of detecting security threats and mitigating them through runtime adaptation techniques [135]. Mature and effective defense mechanisms are proposed to secure a software system [40] [48] [132]. Each of the proposed mechanisms protect a particular vulnerability. For example, malicious clients can:

- send a high number requests per second. The system can issue a puzzle challenge such as CAPTCHA [126] to these requests or blacklist malicious clients [111].
- send requests to access documents which introduce high workload and result in decreasing system performance. Such workload-intensive requests can be mitigated using performance model-driven approaches [8].
- send requests to access documents that contain sensitive information. Such insider attacks can be mitigated by the use of runtime role based access control models that are associated with system resources [7].

A software system can be equipped with defense mechanisms to mitigate all the above-mentioned threats separately. However, the success or failure of each of these attacks is highly dependent on whether the system correctly detects the type of the attack at an early stage. This calls for the need for an intelligent security mechanism. In this chapter, we propose an *Incentive-Based Self-Protection (IBSP)* framework to model a decision-making engine that selects proper countermeasures when facing such scenarios. IBSP does not seek to replace traditional security approaches but rather to introduce a more-collaborative approach to complete them. The benefits of having an IBSP framework are as follows:

- **Early Attack Mitigation**: As soon as malicious behavior is observed, the SPS will take an action while considering the security preferences of the software and malicious intentions of attackers.

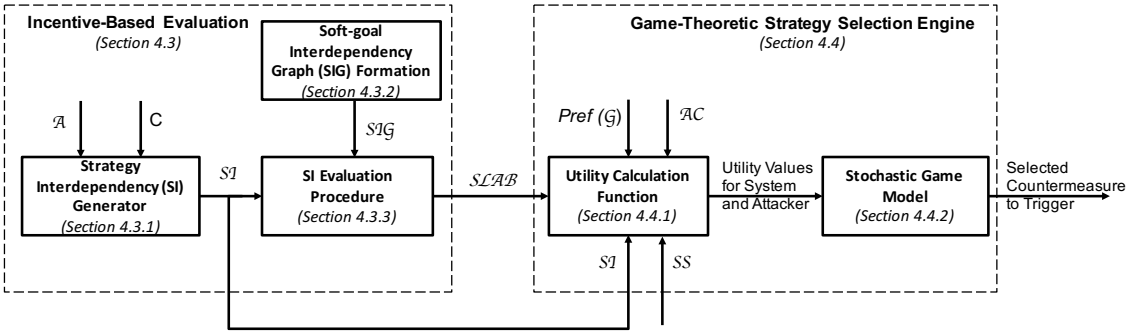


Figure 4.2: The Architecture of Incentive-Based Self-Protection Framework

- **Incorporating Characteristics of Attackers**: By modeling and analyzing intentions of attackers into the decision-making engine, action selection is based on characteristics of attackers as opposed to characteristics of attacks (as is the case in intrusion detection systems).

- **Multi-Objective Decision Making**: In the proposed approach, the interdependencies among strategies are defined with the aid of the SPS system's quality goals. Moreover, incorporating various quality goals is beneficial in terms of capturing the positive/negative impact of a countermeasure on different quality goals. For example, a countermeasure can increase the high performance quality goal while it can be adverse to high throughput quality goal. The decision model uses a utility calculation function in which each quality goal is assigned a weight. Since the utilities are multi-objective, the proposed decision model is multi-objective.

To build an SPS system that supports the IBSP framework, a systematic modeling process should be in place to incorporate quality goals of the SPS system and malicious goals of attackers and relate them to defense/attack strategies. Consequently, the decision model is built upon strategy interdependencies in terms of quality goals. This process is illustrated in Figure 4.2. The architecture of IBSP framework consists of two main phases: (i) Incentive-Based Evaluation, and (ii) Game-Theoretic Strategy Selection Engine. In the first phase, attacks and countermeasures are paired to build SIs. Then, the impact of SIs are evaluated using SIGs [29]. In the second phase, once the labels on quality/malicious goals are generated for all SIs, the utility calculation function generates utility values by considering the quality-goal preferences of stakeholders. Finally, the resulting utility values are input to the stochastic game model, which selects the most proper response in light of the strategy of the

attacker. The formal model of the proposed IBSP framework is provided guided by the following research questions:

- **RQ1.1:** What is the impact of incorporating quality goals of the software system along with the malicious goals of the attacker on the result of the strategy selection engine?
- **RQ1.2:** What is the impact of the system state on the utility of the SPS system?
- **RQ1.3:** What is the impact of the attacker category on the utility of the attacker?

In a software system, when triggering a defense strategy against an attack, not only the fulfillment of quality goals need to be guaranteed, but also the response to the attack must consider the historical data available with regard to the intention of the attacker, category of the attacker, and the state of the system. Each of these questions aims at evaluating the impact of various information on the utility of the system or the attacker. The objective is to *maximize the accumulated utility of the system while minimizing the accumulated utility of the attacker*. The next two sections discuss each of the two phases of the IBSP framework in details.

4.3 Modeling: Incentive-Based Evaluation

For the rest of this chapter, we take the motivating scenario, described in Section 3.1 as a guideline for how the required concepts are captured. The scenario includes two attacks and two mitigations along with six quality and malicious goals. In reality, there will be many types of simultaneous attacks and countermeasures. But in this paper, the objective is to provide a comprehensive explanation of the proposed framework. Hence, a simple scenario helps to demonstrate the required steps in details. The first part of the IBSP framework includes three main processes: (i) Strategy Interdependency Generator, (ii) Soft-Goal Interdependency Graph Formation, and (iii) Strategy Interdependency Evaluation Procedure. The following subsections elaborate on each of the processes.

4.3.1 Strategy Interdependency Generator

Different software systems have different vulnerabilities related to their design and development. First the system vulnerabilities need to be identified. Second, potential attacks that can exploit those vulnerabilities are determined. Third, a set of countermeasures to protect or eliminate the vulnerabilities are defined. For example, in our motivating scenario, two vulnerabilities exist: (i) the ability to service only a limited number of requests in a certain amount of time, and (ii) the existence of sensitive data in the system that require privileged access. Here, we define the attacks and countermeasures for our motivating scenario:

- Set of Attacks: $\mathcal{A} = \{ A_1, A_2 \}$ where A_1 is "*Application-layer DoS Attack*", and A_2 is "*Insider Attack*".
- Set of Countermeasures: $\mathcal{C} = \{ C_1, C_2 \}$ where C_1 is "*Issue Puzzle*", and C_2 is "*Drop Request*".

When the type of an attack is not detected by the SPS system, the decision-making engine faces the dilemma of which strategy to take. The success or failure of an attack depends on which of the available countermeasures is taken against that attack, and similarly the effectiveness of a countermeasure depends on the strategy and the incentive behind an attack. In a nutshell, there is a need to capture the inherent interdependencies among countermeasures and attacks. Based on our motivating scenario, four possible SIs ($\mathcal{SI} = \{SI_1, SI_2, SI_3, SI_4\}$) are defined as shown in Table 4.2.

Table 4.2: Interdependencies of Strategies

Attacks ($\mathcal{A}$) / **Countermeasures** ($\mathcal{C}$)	$\mathbf{A_1}$	$\mathbf{A_2}$
$\mathbf{C_1}$	SI_1	SI_2
$\mathbf{C_2}$	SI_3	SI_4

While both issuing a puzzle challenge and dropping the request can mitigate application-layer DoS attacks, issuing a puzzle challenge (SI_1) can alleviate application-layer DoS attacks more effectively than dropping the incoming malicious request (SI_3). Puzzle challenges can effectively identify malicious users who can hence be blocked, whereas dropping a malicious request does not stop malicious users from

sending requests. These requests cause increased load on the system and consequently raise response times for regular users, diminishing their satisfaction. An insider attacker can successfully pass the puzzle challenge (SI_2) and gain access to the targeted sensitive information, severely compromising a system's quality goals. Insider attacks can be mitigated by not providing a response to malicious requests (dropping the requests – SI_4).

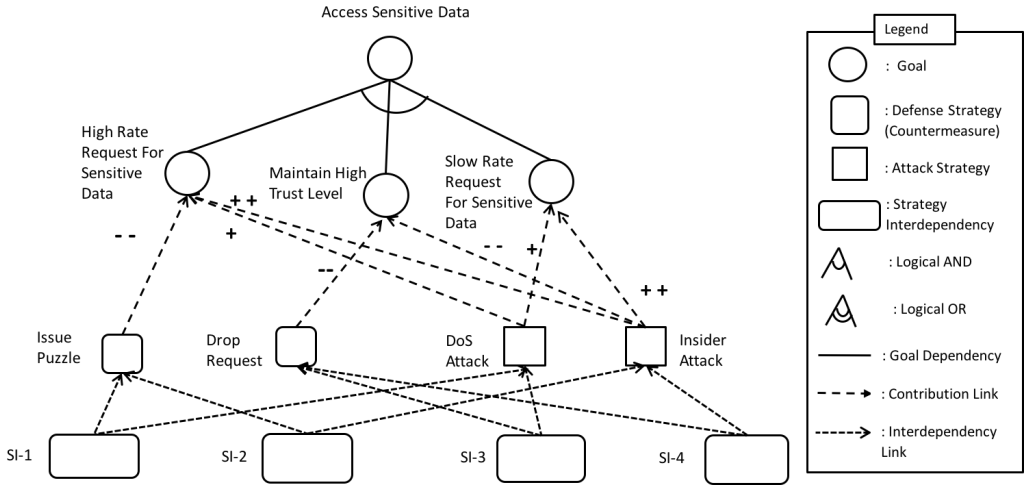
In our motivating scenario (that was described in Chapter 3.1), we have considered all the combinations between attack and defense strategies. In cases that certain combinations do not make scenes, they can still be considered in the model. Later, when we model the impact of strategy interdependencies on the goals, the low/negative impact of such scenarios can be incorporated in the model. Besides, in our motivating scenario, the number of attacks and defense strategies are considered to be two. While we acknowledge that software system may deal with more number of attacks that share similar attack scenarios, in our motivating scenario, we aim at illustrating phases of our proposed framework. IBSP framework has no limitation on the number of attacks and defense strategies.

Modeling strategy interdependencies with the aid of *soft-goals* was proposed in the earlier work at a very high-level belief with the justification for requiring to design a framework that facilitates incorporating soft-goal models at the design-time to decision-making at the runtime [44]. Conceptual modeling of soft-goals supports security trade-off analysis and also provides a mean to measure goal satisfaction quantitatively or qualitatively, which can greatly simplify decision making [38]. Therefore, we employ conceptual modeling of soft-goals by mapping soft-goals to strategy interdependencies.

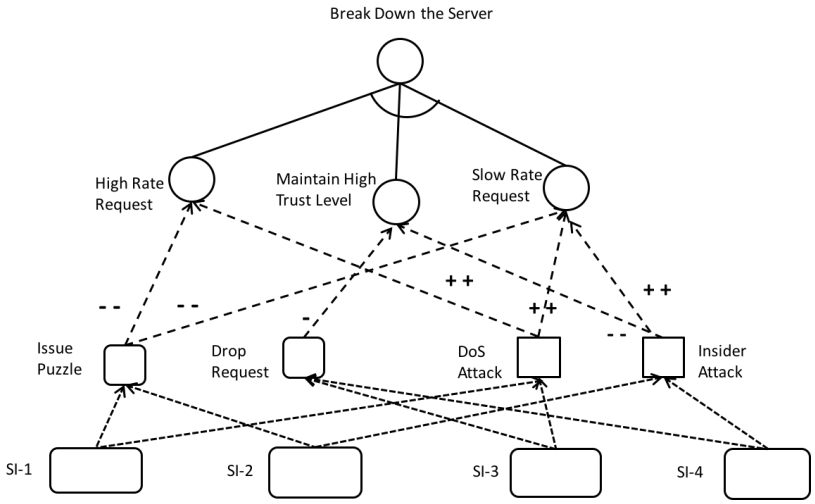
4.3.2 Soft-Goal Interdependency Graph Formation

In our proposed approach, we incorporate satisfaction of quality goals as the criteria on which the SPS system bases its decision. Quality concerns of the SPS system are defined based on the objectives of stakeholders. In our model, stakeholders are either: (i) business owners and administrators, or (ii) end-users. Stakeholders start with articulating high-level goals by specifying the desired behaviors for the system and then decompose these goals into subgoals that can be related to measurable attributes [110]. Quality concerns of the SPS system and malicious goals of attackers in our motivating scenario are summarized in four soft-goals:

- Set of Malicious Goals: $\mathcal{MG} = \{ MG_A , MG_B \}$ where MG_A is "*Access*

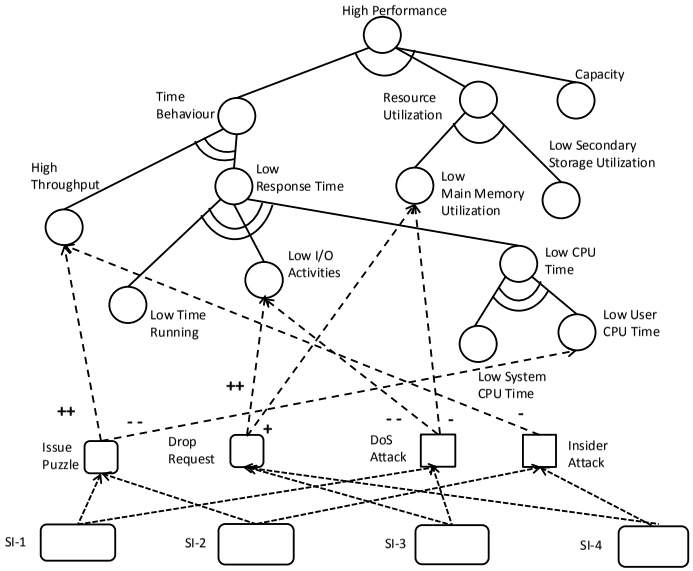


(a) Access Sensitive Data - SIG_1

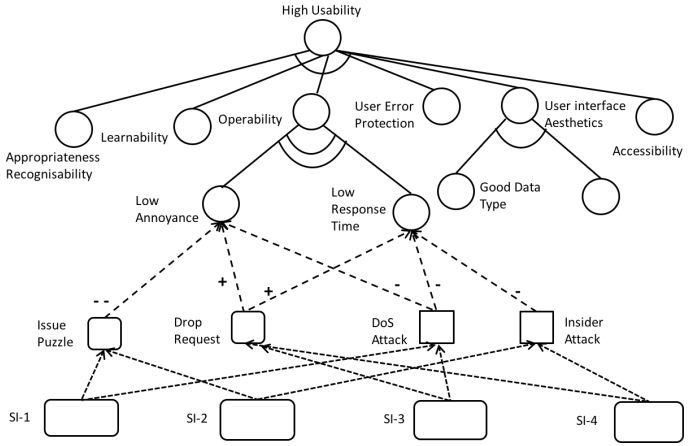


(b) Break Down the Server - SIG_2

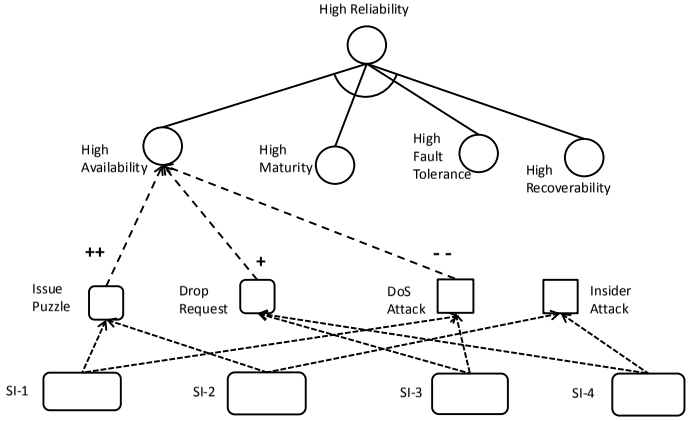
Figure 4.3: Malicious Goals Decomposition Graphs



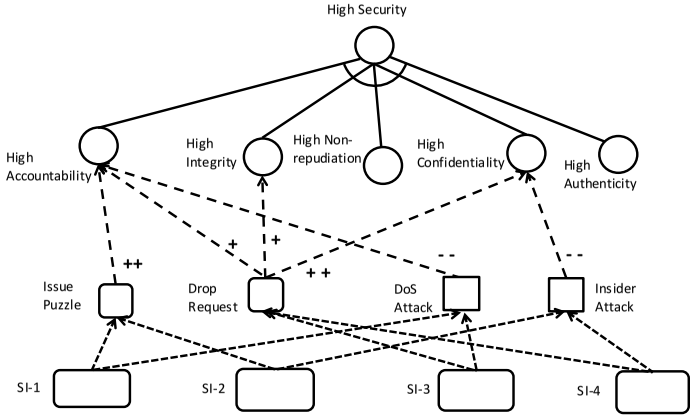
(a) High Performance - SIG_3



(b) High Usability - SIG_4



(c) High Reliability - SIG_5



(d) High Security - SIG_6

Figure 4.4: Quality Goals Decomposition Graphs

Sensitive Data" , and MG_B is "*Break Down the Server*" .

- Set of Quality Goals: $\mathcal{QG} = \{ QG_P, QG_U , QG_R , QG_S \}$ where QG_P is "*High Performance*", QG_U is "*High Usability*" , QG_R is "*High Reliability*", and QG_S is "*High Security*".

In order to model goals and incentives of the software system and the attacker, there are well-studied goal models in the literature that can be employed (e.g., [12] [62] [80] [93] [97] [124]). Any goal model that can model the impact of strategies on goals is desired in our framework. The purpose of engaging a goal model in our framework is twofold: (i) to ease understanding of the impact of attack and defense strategies via visualization, and (ii) to facilitate countermeasure selection based on the satisfaction of goals. In this research work, we do not aim at proposing a brand-new goal model that analysis security aspects of software system. Thus, the incorporated goal model can be substituted with other goal models that support quantitative evaluation of goals (such as goal models in [6] and [55]). The novelty of our framework is to map a design-time goal model to a runtime game-theoretic decision model that captures strategy interdependencies.

To represent information regarding these goals and the strategies that may affect them, we adopt the Non-Functional Requirement (NFR) Framework proposed by Chung et al. [29]. The NFR framework introduces the *Soft-goal Interdependency Graph (SIG)*, which represents each of the goals as a soft-goal and supports systematic modeling of quality/Malicious goals.

In our motivating scenario, the malicious goals of access sensitive data and break down the server are represented in two SIGs ($\mathcal{SIG} = \{SIG_1, SIG_2\}$) in Figure 4.3a and Figure 4.3b. The quality goals of performance, usability, reliability, and security are illustrated in four SIGs ($\mathcal{SIG} = \{SIG_3, SIG_4, SIG_5, SIG_6\}$) in Figure 4.4a, Figure 4.4b, Figure 4.4c, and Figure 4.4d, respectively. The quality goals are refined based on the *International Organization for Standardization/ International Electrotechnical Commission* (ISO/IEC) 25010:2011 [1] guideline. ISO/IEC 25010:2011 introduces product quality model which categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related subcharacteristics [1].

Consequently, each SIG is further refined based on domain knowledge regarding the motivating scenario. This refinement of the performance goal is justified by referring to the work by Tahvildari et al. [120] and the work by Elahi and Yu [38].

The former article considers improving design qualities (such as performance) using transformations, and the latter focuses on analyzing security trade-offs among the competing goals of multiple actors.

The high-level quality goals may denote different meanings for different people (developers, stakeholders, etc.). Hence, it requires that the meaning of each goal be clarified through an iterative process of soft-goal refinement, which involves domain experts. In such refinements, the offspring can contribute fully or partially towards satisfying the goals denoted by the parent goal node. The dependency operators are denoted by AND, OR labels on the arcs. These two contribution operators relate a group of offspring to a parent. In addition, the positive/negative contributions of defense and attack strategies to low-level quality goals can be represented by the aid of SIG in order to investigate the impact of the strategies and their interdependencies on quality goals. Doing so facilitates measurement of the level of satisfaction for each high-level quality goal.

Contribution links between strategies and soft-goals show the impact of actions on soft-goals. As described in Table 4.1, the set of possible impacts include: $\mathcal{IMP} = \{++, +, ?, -, --\}$. Positive/negative contributions that are strong enough to satisfy/deny a soft-goal are represented by *Make*(++)/ *Break*(- -) links. Contributions that are not sufficient to satisfy/deny a soft-goal are represented by *Help*(+)/*Hurt*(-) links. *Unknown* (?) represents situations where the effect of a contribution is unknown.

For example, the performance soft-goal shown in Figure 4.4a is refined into time behaviour, resource utilization, and capacity, where a single arc between edges denotes the AND contribution, meaning that the parent soft-goal is only satisfied/satisficed[1] when all of its offspring are satisfied/satisficed. Each soft-goal can be further decomposed into more detailed soft-goals. For instance, time behavior is broken down into high throughput and low response time with the double arc representing OR relations. Let us assume that the SPS system considers the use of the strategy SI_1, which is "Issue Puzzle" against a "DoS Attack", as discussed by [111]; "Issue Puzzle" strategy (such as CAPTCHA [126]) is one of the potential tactics against DoS attacks if identifying malicious hosts is problematic. Subsequently, the SPS system discovers sooner or later that SI_1 has a positive impact on "Throughput" at the expense of a negative impact on "User CPU Time", as a result of the puzzle challenge. Similarly, the positive and negative contributions of SIs are illustrated in Figure 4.4a, Figure 4.4b, Figure 4.4c, and Figure 4.4d.

[1]Soft goals cannot be clearly evaluated as being satisfied or denied. This is why the term satisficied (originally from Herbert Simon [116]) is used for them [30].

A software engineering concern is to maintain and and update the modeled SIGs. Any change in the modeled goals or strategies need to be applied to the SIGs and hence utility values will get updated accordingly. However, goals and strategies do not change very often.

Using the NFR framework creates a significant step towards filling the gap between quality goals and SIs. Given a defense and attack strategy, one can examine how they affect the desired qualities of the SPS system. In the next subsection, we elaborate on qualitative evaluation of the impact of SIs on quality goals.

4.3.3 Strategy Interdependency Evaluation Procedure

For each SI, the impact of strategies is propagated from offspring to parents in SIG. The impact on high-level quality goals determines the interdependency between the defense and attack strategy in terms of the degree to which high-level quality goals are achieved. Given a SIG, one can decide on the impact of applying an SI on each soft-goal. This is done through the assignment of *labels*. We adopt the qualitative labels used in NFR evaluation [29].

Table 4.3: Propagation Rules Showing Resulting Labels for Contribution Links (Adopted From [61] [121])

Impacts ($\mathcal{IMP}$) / **Labels** ($\mathcal{LAB}$)	**Make** (++)	**Help** (+)	**Unknown** (?)	**Hurt** (-)	**Break** (- -)
Denied ($\times$)	$\times$	W^-	U	W^+	W^+
Weakly Denied (W^-)	W^-	W^-	U	W^+	W^+
Unknown (U)	U	U	U	U	U
Conflict ($\natural$)	$\natural$	$\natural$	U	$\natural$	$\natural$
Weakly Satisfied (W^+)	W^+	W^+	U	W^-	W^-
Satisfied ($\surd$)	$\surd$	W^+	U	W^-	$\times$

In Table 4.3, labels in the set of labels ($\mathcal{LAB} = \{\times, W^-, U, \natural, W^+, \surd\}$) are defined using the notion of satisficeable and deniable. The *Satisfied* ($\surd$)/ *Weakly Satisfied* (W^+) labels represent the presence of evidence which is sufficient/insufficient to satisfy an element. Similarly, *Denied* ($\times$) and *Weakly Denied* (W^-) have the same definition with respect to negative evidence. *Satisfied* ($\surd$) and *Denied* ($\times$) labels are considered as *full labels*. *Conflict* ($\natural$) indicates the presence of both positive and negative evidence of roughly the same strength. *Unknown* (U) represents the

Table 4.4: Cases where Overall Labels can be Automatically Determined (Adopted From [61])

Label Bag Contents	**Resulting Label**
1. The bag has only one label : $\{l\}$	the label: l
2. The bag has multiple full labels ($\surd$ or $\times$) of the same polarity, and no other labels. Examples: $\{\surd, \surd, \surd\}$ or $\{\times, \times\}$	the full label: $\surd$ or $\times$
3. All labels in the bag are of the same polarity, and a full label ($\surd$ or $\times$) is present. Examples: $\{\surd, W^+, \surd\}$ or $\{\times, W^-\}$	the full label: $\surd$ or $\times$
4. The previous human judgment produced $\surd$ or $\times$, and a new contribution is of the same polarity	the full label: $\surd$ or $\times$

situation where there is evidence, but its effect is unknown. To increase flexibility, any soft-goal lacking any label is ignored in the evaluation process.

The labels propagate throughout the SIG using the propagation rules in Table 4.3. Results propagated through contribution links are placed into a "label bag" for that soft-goal. Label bags are manually and in some specific cases automatically (Table 4.4) resolved and produce a single result label. Furthermore, the procedure causes the labels to propagate through AND/OR dependency links by combining them into a single label, and by choosing the minimal/maximal label of the bag. The labels are ordered in increasing order as follows [61]:

$$\times < W^- < U < \natural < W^+ < \surd \tag{4.1}$$

The evaluation procedure is illustrated in Algorithm 1. The algorithm receives two inputs: (i) the set of defined SIs ($\mathcal{SI}$), and (ii) the set of SIGs ($\mathcal{SIG}$) for all the soft goals ($\mathcal{G}$). The output of the algorithm is a set of labels for all SIs and soft-goals ($\mathcal{G}$). The quantitative evaluation procedure starts with assessing initial label values to attack/defense strategies (line 1-3). For each SI, the procedure propagates the impact of strategies in all the defined SIGs. The results of the labels for each soft-goal via each SI are added to a set (lines 4-10). As the labels are propagated through the model links using defined rules, human judgment is needed when multiple conflicting or partial values must be combined, to determine the satisfaction or denial of a soft-goal.

After applying the evaluation procedure, we obtain a set of labels that represent the impact of a particular SI_i on a soft-goals G_j. In Table 4.5, we present the impact of the SIs on the quality and malicious goals which are defined earlier. For

Algorithm 1: Qualitative Evaluation Procedure

Input : $\mathcal{SI}$: A set of defined SIs
Input : $\mathcal{SIG}$: A set of SIGs
Output : $\mathcal{SLAB}$: A set of labels for all SIs and soft-goals

forall SIG_i *in* $\mathcal{SIG}$ **do**
  Initialize labels of SIG_i for each strategy interdependency.
end
forall SI_i *in* $\mathcal{SI}$ **do**
  forall SIG_j *in* $\mathcal{SIG}$ **do**
    while G_j *is not evaluated* **do**
      Propagate SI_i contribution labels from offsprings to parents
    end
    Add $Label(SI_i, G_j)$ (label of goal G_i via strategy interdependency SI_i) to $\mathcal{SLAB}$.
  end
end
return $\mathcal{SLAB}$

Table 4.5: Labels Soft-Goals for SIs

$\mathcal{C}$ \ $\mathcal{A}$	A_1	A_2
C_1	$Label(SI_1, QG_P) = W^+$, $Label(SI_1, QG_U) = \times$ $Label(SI_1, QG_R) = W^+$, $Label(SI_1, QG_S) = \natural$ $Label(SI_1, MG_A) = W^+$, $Label(SI_1, MG_B) = \natural$	$Label(SI_2, QG_P) = \times$, $Label(SI_2, QG_U) = \times$ $Label(SI_2, QG_R) = \times$, $Label(SI_2, QG_S) = \times$ $Label(SI_2, MG_B) = \surd$, $Label(SI_2, MG_B) = W^-$
C_2	$Label(SI_3, QG_P) = \natural$, $Label(SI_3, QG_U) = W^+$ $Label(SI_3, QG_R) = \natural$, $Label(SI_3, QG_R) = W^+$ $Label(SI_3, MG_A) = W^-$, $Label(SI_3, MG_B) = W^+$	$Label(SI_4, QG_P) = W^+$, $Label(SI_4, QG_U) = W^+$ $Label(SI_4, QG_R) = W^+$, $Label(SI_4, QG_S) = W^+$ $Label(SI_4, MG_A) = \natural$, $Label(SI_4, MG_B) = \times$

example, with the aid of Algorithm 1, the impact of SI_1 on the *High Performance* goal (Figure 4.4a) can be evaluated as weakly satisfied ($Label(SI_1, QG_P) = W^+$). Figure 4.4b models *High Usability* soft-goal decomposition. In this SIG, SI_1 evaluates to denying the *High Usability* goal by issuing a puzzle challenge in the face of a DoS attack. SI_1 has impact on *Low Annoyance* and *Low Response Time* soft-goals. The logical OR among all these soft-goals result in a fully denied *High Usability* soft-goal ($Label(SI_1, QG_U) = \times$). Figure 4.4c models *High Reliability* soft-goal in which SI_1 results in making and hurting the *High Availability* via issuing puzzle during DoS attack. Hence, the *High Reliability* goal is considered weakly satisfied. *High Security* quality goal model (Figure 4.4d) illustrates that SI_1 impacts *High Accountability*

both by making and breaking the goal. The impacts are evaluated to conflict in *High Security*. Next, we need to identify the proper strategy among potential strategies that can assist in satisfying the desired quality goals as much as possible.

4.4 Designing: Game-Theoretic Strategy Selection Engine

The second part of the IBSP framework includes two main processes as shown in Figure 4.2: (i) Utility Calculation Function, and (ii) Stochastic Game Model. The former quantifies the impact of strategies on the soft-goals to calculate utility values and the latter maps the utility values to payoffs of a two-player stochastic game in order to find the proper strategy with regard to the equilibrium of the game. In the following, we provide the detail of each process.

4.4.1 Utility Calculation Function

To enable the selection of strategies at runtime, our approach provides trade-off analysis among different strategies by means of utility functions and preferences. The concept of utilities helps to fuse quality goals as well as attacker's incentives into the decision model. We quantify the satisfaction of the soft-goals to values between 0 (low) to 1 (high). Table 4.6 summarizes the satisfaction function ($Sat(Label(SI_i, G_j)) \rightarrow [0..1]$) based on the labels of the quality goals. The function $Label(SI_i, G_j)$ results in a label from the set $\mathcal{LAB}$. Hence the input of the $Sat(Label(SI_i, G_j))$ function is $\mathcal{LAB}$. Similar mapping from qualitative evaluation to qualitative evaluation is defined in [6] by Amyot et al. which can also be used. Both mappings capture positive and negative magnitude of the impacts. They may result in different utility values, but the order of the values won't be affected.

A utility value for an SI represents the satisfaction of the soft-goals that contribute to that SI. Using the $Sat()$ function, we define the utility of the system and the utility of an attacker as follows:

• **System utility**: The preferences of the quality goals is elaborated by assigning a specific weight to each one of them in such a way that the weight value for all the goals sums up to 1. Preferences come from stakeholders' opinions where $\sum_{k=1}^{|\mathcal{QG}|} Pref(QG_k) = 1$. Satisfaction of quality goals for strategy interdependecies ($Sat(Label(SI_i, QG_k))$ is quantified using Table 4.6. By defining a system utility

Table 4.6: Utility Function for a Quality/Malicious Goal

Quality/Malicious Goal Label for G_j via SI_i Label($\mathbf{SI_i, G_j}$) $\rightarrow \mathcal{LAB}$	**Quality/Malicious Goal Satisfaction Sat($\mathcal{LAB}$) $\rightarrow$ [0..1]**
$Label(SI_i, G_j) = \times$	$Sat(\times) = 0$
$Label(SI_i, G_j) = W^-$	$Sat(W^-) = 0.25$
$Label(SI_i, G_j) = U$	$Sat(U) = 0.5$
$Label(SI_i, G_j) = \natural$	$Sat(\natural) = 0$
$Label(SI_i, G_j) = W^+$	$Sat(W^+) = 0.75$
$Label(SI_i, G_j) = \surd$	$Sat(\surd) = 1$

($SUtil(SI_i)$), we measure how "desirable" a SI is for the software system, according to the preferences of the quality goals. The utility of the system is defined as:

$$SUtil(SI_i) = \sum_{k=1}^{|\mathcal{QG}|} Pref(QG_k) * Sat(Label(SI_i, QG_k)) \tag{4.2}$$

• **Attacker utility**: Preferences of the malicious goals of the attacker come from user behavior analysis and the sum of the preferences is 1. Hence, we have: $\sum_{l=1}^{|\mathcal{MG}|} Pref(MG_l) = 1$. Satisfaction of malicious goals for strategy interdependecies ($Sat(Label(SI_i, MG_l))$ is quantified using Table 4.6. The utility of the attacker ($AUtil(SI_i)$) is defined similar to the utility of the system as:

$$AUtil(SI_i) = \sum_{l=1}^{|\mathcal{MG}|} Pref(MG_l) * Sat(Label(SI_i, MG_l)) \tag{4.3}$$

In our example, we have defined 4 SIs, 4 quality goals , and 2 malicious goals. Hence, in Equations 1 and 2 we have: $i = \{1, 2, 3, 4\}$, $k = \{P, U, R, S\}$, and $l = \{A, B\}$. The state of the system can change the impact of an strategy and hence the utility value for the system differs. Similarly, the category of the attacker impacts the strategy of the attacker and accordingly the utility value for the attacker. In our motivating scenario, four system states and four attacker categories are defined:

• **Set of system states**: $\mathcal{SS} = \{\ SS_m\}\ 1 \leq m \leq 4$.

• **Set of attacker categories**: $\mathcal{AC} = \{\ AC_n\}\ 1 \leq n \leq 4$.

A system state (or an attacker category) vary from another system state (or attacker category) in terms of its preferences of the quality goals (or malicious goals). Thus, a particular quality (or malicious) goal's satisfaction value can result in different utility value for the system (or the attacker) depending on the system state (or the attacker category). System state can be configured considering the preferences of the stakeholders over quality goals. Correspondingly, attacker category can be determined by recognizing the desirability of malicious goals for adversaries. Such information can be obtained by looking at the history of attack targeted the software system.

Determining the maliciousness of a user is a challenge by its own and is out of the scope of this research. We can use simplistic behavior monitoring to determine the maliciousness of the action taken by a user. For instance, measures to determine maliciousness of a user can be the amount of traffic generated by the user or the sensitivity of the data that has been accessed by the user. Other approaches can be employed to detect suspicious activities. For example, Lamba et al. [74] describe a model-based approach to cluster sequences of user behaviors within a system and to find suspicious, or anomalous, sequences. Yuan and Maled [134] detect anomalous behavior by mining software component interactions from system execution history.

4.4.2 Stochastic Game Model

Competitive behavior of the software system and the attacker over promoting/degrading soft-goals is a natural fit for modeling a game between two players. While other approaches in the literature consider maximizing the expected utility of the software system (e.g., [111]), the application of game theory provides the ability to find the maximum utility of the software system while acknowledging the possible strategies of the attacker. As a result the selected strategy is the strategy that maximizes the utility of the software system while minimizing the utility of the attacker.

Having the utility of the software system and the attacker, the defined two-player normal form game is represented in Table 4.7 in general format. For a particular software system, utility values can be calculated considering the system states and attacker category as described in the previous subsection. By populating the modeled game in Table 4.7 with the actual utility values, the solution of the game (Nash Equilibrium [78]) can be calculated. Afterwards, the defense strategy (countermeasure) is selected according to the Nash Equilibrium.

In game theory, a stochastic game is a dynamic game that is played in a sequence of stages. The players select actions and each player receives a payoff that depends

Table 4.7: Normal Form Game Model

Attacks ($\mathcal{A}$) / **Countermeasures** ($\mathcal{C}$)	$\boldsymbol{A_1}$	$\boldsymbol{A_2}$
$\boldsymbol{C_1}$	$SUtil(SI_1), AUtil(SI_1)$	$SUtil(SI_2), AUtil(SI_2)$
$\boldsymbol{C_2}$	$SUtil(SI_3), AUtil(SI_3)$	$SUtil(SI_4), AUtil(SI_4)$

on the current state and the chosen actions. In our designed strategy selection engine, the payoff values are calculated at each play depending on the state of the system. Hence, stochastic game technique fits our modeled game and we employ this technique in our dynamic game model to solve the game. Moreover, techniques such as probabilistic model checking provide a means to model and analyze systems that demonstrate stochastic behavior and enables reasoning quantitatively about strategy selection. In the next section, we formally model the proposed IBSP framework with the aid of Stochastic Multiplayer Games (SMGs).

4.5 Realizing: Formal Modeling via Stochastic Multiplayer Game

We argue that considering the intentions/goals of attackers can enhance the effectivness of selecting a defense strategy. To reason about the impact of incorporating such information into the proposed strategy selection engine, a formalization of the decision model as stochastic multiplayer games is used which is based on probabilistic model checking in formal models [23]. Our modeling approach for game-theoretic strategy selection engine in IBSP framework is based on defining a two-player game in which one player is the SPS system and the other player is the attacker. The goal of the system player is to maximize its accumulated utility during the runtime of the system[2]. In the remainder of this section, we first introduce some background on the formal SMG modeling technique that we employed. Next, we provide a detailed description of our SMG model implemented in the probabilistic model-checker tool called PRISM-game [24].

[2]An implementation of IBSP technique is available at https://github.com/mahsa-emamitaba/IBSP-PRISM

4.5.1 Model Checking Stochastic Multiplayer Games

Automatic verification techniques for the modeling and analysis of probabilistic systems provide a means to enable quantitative reasoning about the probability and rewards of the system. The systems that incorporate competitive behavior can be modelled as turn-based SMGs [23]. The logic of this technique for expressing quantitative properties of stochastic multi-player games is called rPTAL. It is the extension of the logic PATL [25]. PATL is itself a probabilistic extension of ATL [5], a widely used logic for reasoning about multi-player games and multi-agent systems. The PRISM-game tool [24] provides a high-level language for modeling SMGs and implements rPATL model checking for their analysis. Reasoning strategies based on probabilistic model checking of Stochastic Multiplayer Games (SMGs) has been a subject of recent interest. SMG is employed in analyzing latency-aware proactive adaptation [16] and human participation in self-adaptive systems [17]. The authors of these papers model a stochastic game to analyze their proposed technique and compare it with different approaches.

Similar to the SMG analysis in this section, the interplay between a self-adaptive system and a potentially adversarial environment as an SMG, and the analysis that accounts for the strategy of the adversary when selecting counter-measures has been described in [111] [16] [17] [14] [18] [112] [13] [15] [19]. In our SMG modeling, we borrow concepts explained in these SMG modelings to formulate the utilities and define the players. We extend their modelings in the following ways:

- In our modeling, the impact of actions are defined by incorporating strategy interdependency which consists of actions of both players (the software system and the attacker). In previous SMG modelings (e.g., in work by Schmerl et al. [111]), the impact caused by the actions of only one player (the software system) is considered in the decision model.

- In our modeling, we build the utility functions based on high-level soft-goals. These soft-goals are defined using SIG and consequently are broken down to lower-level goals that are measurable with metrics. In previous modelings (e.g., in work by Schmerl et al. [111]) the quality objectives are directly quantified using utility functions.

- In our modeling, we consider the utility of attacker to analyze the decision-model. In previous modelings (e.g., in work by Cámara et al. [17]), only the utility of the software system is defined while the utility of the other player is

not part of the investigation. When SPS system is uncertain about the attack-type, incorporating attacker utility helps the system to select more effective countermeasures which minimizes the utility of the attacker while maximizes the utility of the system.

In the next subsection, we exploit SMG to provide a formal modeling of our proposed IBSP framework. To compare IBSP with alternative approaches, we model four decision-making approaches, namely: (i) IBSP mitigation, (ii) Random mitigation, (iii) Fixed-Drop mitigation, and (iv) Fixed-Puzzle mitigation. Random mitigation randomly selects a countermeasure while Fixed-Drop mitigation and Fixed-Puzzle mitigation select a fixed countermeasure (which is either dropping the request or issuing a puzzle challenge).

4.5.2 Formal SMG Model of IBSP Mitigation Approach

For the rest of this section, we realize the proposed framework by implementing the described attack scenario (in Section 3.1) via SMG formal modeling. As shown in Figure 4.2, IBSP framework consists of five main processes. This section describes how each process is executed in our IBSP formal modeling preparation.

• **Strategy Interdependency Generator**: The two players of the game (SPS system and attacker) are modeled via SMG *players* and *modules*. Afterwards, the actions of players and SIs that are described in Table 4.2 in Section 4.3.1 can be modeled via SMG *actions*. Each SI is modeled with the pre-condition specifying certain actions of players that are combined as SI . For example, SI_1 is modeled so that only gets executed if the action of system player is *Issuing Puzzle* and the action of the attacker player is *Application-layer DoS Attack*.

• **Soft-Goal Interdependency Graph Formation**: As explained in Section 4.3.2, we define two high-level soft-goals for the software system and two high-level soft-goals for the attacker. These goals are modeled using SIG as illustrated in Figure 4.4. Consequently these four soft-goals can be modeled via SMG *variables* and by defining lower and upper limits for each variable.

• **Strategy Interdependency Evaluation Procedure**: The impact of the defined SIs are evaluated using the proposed procedure in Section 4.3.3. The result of evaluation is illustrated in Table 4.5. These negative/positive impacts on the defined goals can be modeled via SMG *formulas*. Each goal has a formula that embodies the negative/positive impact on the goal's satisfaction by subtracting/adding from/to the value of the goal's variable.

The game defined in our modeling is a turn-based game between two players: the software system and the attacker. Listing 4.1 defines the two players in the SMG. Player `system` is in control of asynchronous actions that the software system can take. These action are defined later in the `software` module. In our formal model, depending on the two types of attacks and the two countermeasures, four pair of strategies (strategy interdependency) are possible ($C1\_A1$, $C1\_A2$, $C2\_A1$, and $C2\_A2$). For each pair off strategies, we formulate the effect of the actual attack and its corresponding countermeasure on the attributes.

```
1  player system
2  software, [C1_A1],[C1_A2],[C2_A1],[C2_A2]
3  endplayer
4  player attacker
5  attacker, [DOS], [INSIDER]
6  endplayer
7  const SYS_TURN = 1, ATTACKER_TURN = 2;
8  global turn: [SYS_TURN..ATTACKER_TURN] init SYS_TURN;
9  global t : [0..MAX_TIME] init 0;
10 const MAX_P = 100;
11 const MAX_U = 100;
12 const MAX_R = 100;
13 const MAX_S = 100;
14 const MAX_A = 100;
15 const MAX_B = 100;
```

Listing 4.1: Players Definition in SMG Model

Player `attacker` controls malicious requests that are sent to the software. These requests are defined in the `attacker` module. In line 8, the global variable `turn` is used to ensure that players take actions alternatively. The global variable in line 9 defines the time frame for the game continuation in the model. Each time a player takes an action, the value of `t` increments one unit. Both players check whether the end of execution (`MAX_TIME`) has been reached or not before taking an action.

```
1  // Case [C1–A1] Puzzle – DoS
2  formula C1_A1__QG_P = 75;
3  formula C1_A1__QG_U = 0;
4  formula C1_A1__QG_R = 100;
5  formula C1_A1__QG_S = 100;
6  formula C1_A1__MG_A = 50;
7  formula C1_A1__MG_B = 0;
8
9  // Case [C2–A1] Drop – DoS
10 formula C1_A2__QG_P = 75;
11 formula C1_A2__QG_U = 0;
12 formula C1_A2__QG_R = 25;
13 formula C1_A2__QG_S = 0;
14 formula C1_A2__MG_A = 100;
15 formula C1_A2__MG_B = 50;
16 ...
```

Listing 4.2: Impact of SIs on Attributes

Listing 4.2 formulates the impact of strategy pairs on the quality goals. For example, lines 2-7 formulate the impact on when taking an issue-puzzle-challenge strategy against an application-layer DoS attack strategy ($C1\_A1$). Here, $C1\_A1\_\_QG\_P$, $C1\_A1\_\_QG\_U$, $C1\_A1\_\_QG\_R$, $C1\_A1\_\_QG\_S$, $C1\_A1\_\_MG\_A$, and $C1\_A1\_\_MG\_B$ represent the impact of strategy $C1\_A1$ on the following goals: performance, usability, reliability, security, accessing sensitive data, and breaking down the system. For example, in line 4, the impact on reliability goal is 75 as the goal will be weakly satisfied via this strategy. The rest of the strategy interdependencies are formulated accordingly.

• **Utility Calculation Function**: Utilities for SIs are defined as described in Section 4.4.1. For each of the four quality/malicious goals, we have assigned a weight in order to calculate the utilities. The utilities for each player can be modeled via SMG *rewards*. To assign weights to the quality goals, we consider the state of the software system. The rationale for doing so is that the preferences of quality goals may vary depending on the state of the software system. For example, *high performance* is an essential quality goal when a system is down, but in other states, *high usability* has a higher priority. Table 4.8 illustrates the preferences/weights of the quality goals the defined states of the system.

Table 4.8: Defining Preferences for Each System State

System State / **Preferences**	SS_1	SS_2	SS_3	SS_4
$Pref(QG_P)$	0.70	0.10	0.10	0.10
$Pref(QG_U)$	0.10	0.70	0.10	0.10
$Pref(QG_R)$	0.10	0.10	0.70	0.10
$Pref(QG_S)$	0.10	0.10	0.10	0.70

Each attacker, depending on its malicious intention, has different strategies. Therefore, the malicious goals' preferences for each category of attacker differ from each other. Characterizing these categories helps to identify attackers' objectives and consequently the likelihood of certain attacks threatening a system at any point of time. In Table 4.9, the preferences of attackers in our running example are defined.

Utility functions are essential to enabling quantitative decision making. In our SMG model, we also formulate utility functions for each quality goal. When the system takes the correct/incorrect defense strategy, it results in increasing/decreasing satisfaction of these goals. The total utility of the software system and the attacker

Table 4.9: Defining Preferences for Each Category of Attacker

Attacker Category / **Preferences**	AC_1	AC_2	AC_3	AC_4
$Pref(MG_A)$	0.50	0.90	0.75	0.10
$Pref(MG_B)$	0.50	0.10	0.25	0.90

are formulated as rewards that are gained each time the system takes an action in its turn.

```
1  formula SAT_QG_P = QG_P/MAX_P;
2  formula SAT_QG_U = QG_U/MAX_U;
3  formula SAT_QG_R = QG_R/MAX_R;
4  formula SAT_QG_S = QG_S/MAX_S;
5
6  // System State : SS1
7  const Pref_QG_P = 0.70;
8  const Pref_QG_U = 0.10;
9  const Pref_QG_R = 0.10;
10 const Pref_QG_S = 0.10;
11 ...
12
13 // System utility
14 rewards "sys_util"
15 (turn = SYS_TURN) : (Pref_QG_P*SAT_QG_P + Pref_QG_U*SAT_QG_U + Pref_QG_R*
       SAT_QG_R + Pref_QG_S*SAT_QG_S);
16 endrewards
17
18 formula SAT_MG_A = MG_A/MAX_A;
19 formula SAT_MG_B = MG_B/MAX_B;
20
21 // Attacker Category: AC1
22 const Pref_MG_A = 0.50;
23 const Pref_MG_B = 0.50;
24 ...
25
26 //Attacker utiliry
27 rewards "att_util"
28 (turn = SYS_TURN) : (Pref_MG_A*SAT_MG_A + Pref_MG_B*SAT_MG_B) ;
29 endrewards
```

Listing 4.3: Utility Reward Structure for the SPS System

The utilities of the software system and the attacker are defined based on the impact of strategies on the four quality goals and the two malicious goals. Listing 4.3 illustrates encoding of utility functions for each quality goal. Lines 1-4 formulate the satisfaction of quality goals. Lines 7-10 define the preferences of goals for *SS1*. Lines 14-16 define the reward structure for the system (`sys_util`) based on the preferences defined for each quality goal (which will be varied based on the system state). Lines

27-29 define the reward structure for the attacker (`att_util`) based on the preferences defined for each malicious goal (which will be varied based on the attacker category).

• **Stochastic Game Model**: As explained in Section 4.4.2, the decision-making engine is modeled as a game-theoretic strategy selection engine which is based on stochastic game modeling. When the SMG model is running, both players (SPS system and attacker) play an strategy without knowing the strategy of the other player. The SMG model runs for certain time as specified in the modeling. During the running of the model each player selects an strategy one at a time.

```
1  module software
2  QG_P : [0..MAX_P] init INIT_P;
3  QG_U : [0..MAX_U] init INIT_U;
4  QG_R : [0..MAX_R] init INIT_R;
5  QG_S : [0..MAX_S] init INIT_S;
6  MG_A : [0..MAX_A] init INIT_A;
7  MG_B : [0..MAX_B] init INIT_B;
8
9  countermeasure_selected: bool init false;
10
11 // Game-Theretic Mitigation
12 [] (turn = SYS_TURN)& (t<MAX_TIME) & (!countermeasure_selected)->
13 (sys_action'=C1) & (countermeasure_selected'= true) ;
14 [] (turn = SYS_TURN)& (t<MAX_TIME) & (!countermeasure_selected)->
15 (sys_action'=C2) & (countermeasure_selected'= true) ;
16
17 // Update utilities based on both players actions
18 [C1_A1] (turn = SYS_TURN)& (t<MAX_TIME)& (sys_action =C1) & (user_action=A1) & (
       countermeasure_selected)->
19 (t'= t+1) & (turn'= ATTACKER_TURN) & (countermeasure_selected'= false) &
20 (QG_P'=C1_A1__QG_P) & (QG_U'=C1_A1__QG_U) & (QG_R'=C1_A1__QG_R) & (QG_S'=
       C1_A1__QG_S) &  (MG_A'=C1_A1__MG_A) & (MG_B'=C1_A1__MG_B) ;
21
22 [C1_A2] (turn = SYS_TURN)& (t<MAX_TIME)& (sys_action =C1) & (user_action=A2) & (
       countermeasure_selected)->
23 (t'= t+1) & (turn'= ATTACKER_TURN) & (countermeasure_selected'= false) &
24 (QG_P'=C1_A2__QG_P) & (QG_U'=C1_A2__QG_U) & (QG_R'=C1_A2__QG_R) & (QG_S'=
       C1_A2__QG_S) &  (MG_A'=C1_A2__MG_A) & (MG_B'=C1_A2__MG_B) ;
25
26 [C2_A1] (turn = SYS_TURN)& (t<MAX_TIME)& (sys_action =C2) & (user_action=A1) & (
       countermeasure_selected)->
27 (t'= t+1) & (turn'= ATTACKER_TURN) & (countermeasure_selected'= false) &
28 (QG_P'=C2_A1__QG_P) & (QG_U'=C2_A1__QG_U) & (QG_R'=C2_A1__QG_R) & (QG_S'=
       C2_A1__QG_S) & (MG_A'=C2_A1__MG_A) & (MG_B'=C2_A1__MG_B) ;
29
30 [C2_A2] (turn = SYS_TURN)& (t<MAX_TIME)& (sys_action =C2) & (user_action=A2) & (
       countermeasure_selected)->
31 (t'= t+1) & (turn'= ATTACKER_TURN) & (countermeasure_selected'= false) &
32 (QG_P'=C2_A2__QG_P) & (QG_U'=C2_A2__QG_U) & (QG_R'=C2_A2__QG_R) & (QG_S'=
       C2_A2__QG_S) & (MG_A'=C2_A2__MG_A) & (MG_B'=C2_A2__MG_B) ;
33
34 endmodule
```

Listing 4.4: Software Module with IBSP

At each play, the SPS system may either take *Drop-Request* or *Issue-Puzzle*. Similarly, the attacker may either take *Application-layer DoS Attack* or *Insider Attack* strategy. The strategy of the players could be different than their previous strategy. The stochastic two-player game that is modeled via SMG can be analyzed in terms of the accumulated value of utilities throughout the execution.

The software system's defense strategy is described in the module `software` in Listing 4.4. The effect of countermeasures on quality goals are captured by updating the satisfaction level of quality goals based on the formulas in Listing 4.2 for the four strategy pairs that are defined. For example, in lines 17-19, the strategy of the software system works for application-layer DoS attacks by issuing a puzzle challenge ($C1\_A1$). Lines 21-23, 25-27, and 29-31 represent $C2\_A1$, $C1\_A2$, and $C2\_A2$ respectively. Each play results in a utility (*reward*) value and at the end of the experiments the accumulated utilities are calculated. The model can be analyzed in terms of the accumulated value of utilities throughout the execution.

The scalability of the modeled SMG is limited by PRISM-games. The size of the SMG model can grow with increasing the number of goals and strategies. The current version of the tool can handle models up to 10^7 states. Our implemented model has 877 states. To evaluate the modeled IBSP mitigation approach compare to other alternatives, we model three mitigation approaches as described in the following subsection.

4.5.3 Formal SMG Model of Random, Fixed-Drop, and Fixed-Puzzle Mitigation Approaches

Random mitigation consists of randomly selecting a countermeasure (from the set of available countermeasures) as the type of attack (either application-layer DoS attack or insider attack) is not certain. In our SMG modeling of random mitigation, the player SPS system can be modeled in such a way that the probability of the SPS system issuing a puzzle challenge is 50%, and the probability of it dropping requests is 50%. Hence, each countermeasure is selected randomly with the same probability.

In Listing 4.5, the module `software` is enriched with a random mitigation approach. Since there are two possible countermeasures to select from, the probability of the system issuing a puzzle challenge is 50% (line 5), and the probability of it dropping requests is 50% (line 6).

```
1  module software
2  countermeasure_selected: bool init false;
3
4  [] (turn = SYS_TURN)& (t<MAX_TIME) & (!countermeasure_selected)->
5  0.5: (sys_action'=C1) & (countermeasure_selected'= true) +
6  0.5: (sys_action'=C2) & (countermeasure_selected'= true) ;
7
8  // Update utilities based on both players actions
9  ...
10 endmodule
```

Listing 4.5: Software Module with Random Mitigation

When the type of attack is uncertain, another possible mitigation approach is to choose a fixed countermeasure without considering the uncertainty in the type of the attack. Hence, the fixed countermeasure either mitigates the attack successfully or fails to mitigate the attack depending on the attack. We modeled two fixed mitigation approaches: (i) *Fixed-Drop* which always drops the suspicious request, and (ii) *Fixed-Puzzle* which always issues a puzzle challenge to the malicious user. countermeasure without considering the strategy of the attacker. In Listing 4.6, the module `software` is modeled to always select the countermeasure *DROP* which drops any suspicious request.

```
1  module software
2
3  [] (turn = SYS_TURN)& (t<MAX_TIME) & (!countermeasure_selected)->
4  (sys_action'=C2) & (countermeasure_selected'= true);
5
6  // Update utilities based on both players actions
7  ...
8  endmodule
```

Listing 4.6: Software Module with Fixed Mitigation

So far, four mitigation approaches that we modeled via formal SMG model are described. Hence, four stochastic games are modeled using PRISM-games. In each of these games the system player employs one of the four mitigation approaches. Each game also has a second player which is an intelligent attacker that sends malicious requests that could result in either application-layer DoS attack or insider attack. The attacker's SMG model (described in Section 4.5.4) is the same in all the four modeled games. The modeled attacker targets software systems by changing its strategy in response to a software system's strategy. The attacker either issue a DoS or an insider attack at each run (50 times in our experiments). Each strategy results in a different impact on the defined quality/malicious goals. This impact is modeled via SMG *formulas*.

4.5.4 Attacker Formal SMG Model

In the following, attacker player is modeled using the formal SMG model. The modeled attacker targets software systems by changing its strategy in response to a software system's strategy. Listing 4.7, shows the encoding for triggering malicious requests and then yielding the turn to the `system` player.

```
1  module attacker
2  attack_type_selected : bool init false;
3
4  // Attacker playes the game
5  [] (turn=ATTACKER_TURN) & (t<MAX_TIME) & (!attack_type_selected)
6  -> (user_action'=A1) & (attack_type_selected'=true);
7  [] (turn=ATTACKER_TURN) & (t<MAX_TIME) & (!attack_type_selected)
8  -> (user_action'=A2) & (attack_type_selected'=true);
9
10 [DOS] (turn=ATTACKER_TURN) & (t<MAX_TIME) & (user_action=A1) & (
       attack_type_selected)
11 -> (t'= t+1)&(turn'=SYS_TURN)& (attack_type_selected'=false);
12 [INSIDER] (turn=ATTACKER_TURN) & (t<MAX_TIME) & (user_action=A2) & (
       attack_type_selected)
13 -> (t'= t+1)&(turn'=SYS_TURN)& (attack_type_selected'=false);
14
15 endmodule
```

Listing 4.7: User Module Modeling a Strategy-Aware Attacker

In this section, we modeled the interactions between the SPS system and the attacker as an stochastic game using PRISM-games. In the next section, we provide analyses of the modeled mitigation approaches in response to the modeled attacker.

4.6 Analyzing IBSP via Stochastic Multiplayer Game

Model checking in PRISM-game [24] is specified in the logic rPATL [23], which combines elements of the probabilistic temporal logic PCTL for Markov Decision Processes (MDPs), the logic ATL [5] for games and agent-based systems, and extensions of PRISM's reward operators. In order to evaluate IBSP in PRISM, we analyze the maximum utility that a mitigation approach can *guarantee*, independently of the behavior of the attacker (real-guarantee scenario). Such analysis via PRISM is also employed by Cámara et al. [16] where authors also consider the *best-case scenario*. However, in the modeled two-player game, the players will not play in favor of each other and for this reason we omitted analysis on the best-case scenarios.

Our SMG models runs for the specified time frame which is 50 in our experiments. This means that the system is under attack for 50 times and selects a countermeasure

for 50 times. Hence, 50 *SIs* occur during runtime of the system. For each SI, the utility of the system and the attacker is calculated. The accumulated utility of the system/attacker is the sum of all the utilities (for the system/attacker) that are gained via applying the selected countermeasures against an attack during the runtime of the system (Equation 4.4 and Equation 4.5).

$$AUS_{Name} = \sum_{j=1}^{n} SUtil(SI_i) \tag{4.4}$$

$$AUA_{Name} = \sum_{j=1}^{n} AUtil(SI_i) \tag{4.5}$$

Tables 4.10, and 4.11 present the result for the accumulated utility calculated for the system and the attacker by four different mitigation types: (i) *IBSP*, (ii) *Random*, (iii) *Fixed-Drop*, and (iv) *Fixed-Puzzle*. To understand whether the proposed incentive-based approach is a suitable solution, we compare the results of IBSP to those of the three other techniques. We define the delta between the *system* utility accumulated by the IBSP mitigation (AUS_{IBSP}) and the random mitigation (AUS_{Random}) as ΔAUS_R (Subscript R stands for Random):

$$\Delta AUS_R = (1 - \frac{AUS_{Random}}{AUS_{IBSP}}) * 100 \tag{4.6}$$

Similarly, the delta between the *attacker* utility accumulated by the IBSP mitigation (AUA_{IBSP}) and the random mitigation (AUA_{Random}) as ΔAUA_R (Subscript R stands for Random):

$$\Delta AUA_R = (1 - \frac{AUA_{Random}}{AUA_{IBSP}}) * 100 \tag{4.7}$$

Moreover, the delta between the *system* utility accumulated by the IBSP mitigation (AUS_{IBSP}) and the fixed-drop mitigation ($AUS_{Fixed-Drop}$) is computed as ΔAUS_{FD} (Subscript FD stands for Fixed-Drop):

$$\Delta AUS_{FD} = (1 - \frac{AUS_{Fixed-Drop}}{AUS_{IBSP}}) * 100 \tag{4.8}$$

Similar to the above equation, the delta between the *attacker* utility accumulated by the IBSP mitigation (AUA_{IBSP}) and the fixed-drop mitigation ($AUA_{Fixed-Drop}$) is computed as ΔAUA_{FD} (Subscript FD stands for Fixed-Drop):

$$\Delta AUA_{FD} = (1 - \frac{AUA_{Fixed-Drop}}{AUA_{IBSP}}) * 100 \tag{4.9}$$

Finally, we define the delta between the system utility accumulated by the IBSP mitigation (AUS_{IBSP}) and the fixed-puzzle ($AUS_{Fixed-Puzzle}$) mitigation as ΔAUS_{FP} (Subscript FP stands for Fixed-Puzzle):

$$\Delta AUS_{FP} = (1 - \frac{AUS_{Fixed-Puzzle}}{AUS_{IBSP}}) * 100 \tag{4.10}$$

For the case of attacker, the above equation changes to ΔAUA_{FP} (Subscript FP stands for Fixed-Puzzle):

$$\Delta AUA_{FP} = (1 - \frac{AUA_{Fixed-Puzzle}}{AUA_{IBSP}}) * 100 \tag{4.11}$$

These values are used to analyze and compare various approaches in the modeled SMG. The objective is to study the effectiveness of IBSP in face of an intelligent attacker. We assume that the attacker dynamically changes its strategy by considering the defense strategy taken by the SPS system.

4.6.1 Obtained Results

The rest of this section reports the results of our analysis (presented in Table 4.10 and Table 4.11), whose objective is to answer the following research questions:

- **RQ1.1:** What is the impact of incorporating quality goals of the software system along with the malicious goals of the attacker on the result of the strategy selection engine?

  To answer RQ1.1, IBSP mitigation is compared with three other mitigation approaches that do not consider incorporation of soft-goals in their decision model. Table 4.10 shows that IBSP mitigation either outperforms the other three mitigation approaches, or results in the same accrued utility value. The IBSP approach results in 0% to 86% higher accumulated utility than that of alternative approaches.

  The accumulated utilities of the attacker with different categories are shown in Table 4.11. We observe that $\Delta AUA_R, \Delta AUA_{FD}$, and ΔAUA_{FP} all have

negative values. This implies the effectiveness of IBSP in terms of causing a lower utility for the attacker (response to RQ1.1). This is due to the fact that the countermeasure selection is done with the incorporation of attacker's strategy and malicious goal.

Table 4.10: The Accumulated Utility of the SPS System

Strategy / System State	AUS_{IBSP}	AUS_{Random}	$\Delta AUS_{R}(\%)$	$AUS_{Fixed-Drop}$	$\Delta AUS_{FD}(\%)$	$AUS_{Fixed-Puzzle}$	$\Delta AUS_{FP}(\%)$
SS1	**27.4**	20.8	24%	14.2	48%	27.4	0%
SS2	**35.8**	20.8	41%	**35.8**	0%	5.7	83%
SS3	**43**	25.59	40%	8.2	80%	13	69%
SS4	**43**	24.4	43%	15.4	64%	5.79	86%

Table 4.11: The Accumulated Utility of the Attacker

Strategy / Attacker Category	AUA_{IBSP}	AUA_{Random}	$\Delta AUA_{R}(\%)$	$AUA_{Fixed-Drop}$	$\Delta AUA_{FD}(\%)$	$AUA_{Fixed-Puzzle}$	$\Delta AUA_{FP}(\%)$
AC1	**13**	25	-92%	37	-184%	37	-184%
AC2	**22.59**	25	-10%	27.4	-21%	46.6	-106%
AC3	**19**	25	-31%	31	-63%	43	-126%
AC4	**22.59**	25	-10%	46.6	-106%	27.4	-21%

- **RQ1.2:** What is the impact of the system state on the utility of the SPS system?

  Regarding research question RQ1.2, as illustrated in Table 4.10, accumulated utility values for all four strategies varies depending on the preferences of the quality goals. While IBSP outperforms other alternatives, Random mitigation results in both higher and lower accumulated utility values compare to fixed-puzzle and fixed-drop mitigation approaches (in *SS1* and *SS2*). In *SS3* and *SS4*, Random mitigation beats both fixed mitigation approaches. In all four system states, Ransom mitigation results in very similar accumulated utility values. Hence, the preferences defined in systems states has the lowest impact on the outcome of the Random mitigation.

  Therefore, in response to RQ1.2, the state of the system has positive/negative impact on the utility of the system if the selected countermeasure causes positive/negative contribution in satisfaction of a particular quality goal which has a major contribution in the state of the system.

- **RQ1.3:** What is the impact of the attacker category on the utility of the attacker?

  To answer research question RQ1.3, we compare the accumulated utility values in Table 4.11 for the four attacker categories. IBSP produces the lowest accumulated utility values compare to its alternative mitigation approaches. Random mitigation results in the same values in all four cases of attacker categories. Comparing Random mitigation with fixed mitigation approaches, in all of the categories, it resulted in lower utility outcome. Hence, after IBSP, Random mitigation is a more attractive mitigation approach than fixed-puzzle and fixed-drop approaches.

  Based on these observations, attacker category can have positive/negative impact on the utility of the attacker without regard to the mitigation strategy (addressing RQ1.3).

New security challenges have emerged with the exponential growth of software systems and various threats and attacks have been introduced [63]. Hence, a specific defense strategy that is capable of addressing diverse types of attacks cannot be developed. Thus, a coordination among different defense strategies is required to be able to select and trigger the more proper one. In our proposed decision-making engine we aim at providing this coordination via incorporating various information from the system (quality goals and states) and from the attacker (malicious goals and categories). The result illustrate that using IBSP, the SPS system gains higher accumulated utility because it considers what could be the action/strategy taken by the attacker. As a result the SPS system becomes more intelligent when it faces attacks that seem to be similar in their scenario while they are different in their malicious goal. Our results at this stage indicate that the IBSP framework can improve the quality of decision-making when facing various types of attacks.

4.6.2 Threats to Validity

The research presented in this chapter is intended to provide a framework for decision-making in SPS systems. IBSP framework is augmented with the incorporation of the soft-goals which include both quality goals and malicious goals. Integrating the intentions of attackers into the decision model helps to broaden the diverse combinations of countermeasures when facing with uncertainty in the type of the attack. The decision-making engine of the IBSP framework is modeled with the aid of the formal

model checking of SMG model. It is important to describe the possible threats to validity as well as ways to prevent their occurrence.

There could be several factors that influence the results of the our studies. Therefore, we cover threats to validity for the proposed IBSP framework along with the provided formal modeling of the IBSP framework via SMG modeling. IBSP framework has two main components as mentioned in Section 4.2: Incentive-Based Evaluation (described in Section 4.3), and Game-Theoretic Strategy Selection Engine (explained in Section 4.4). Besides, threats to validity of the modeled IBSP via SMG (detailed in Section 4.5) is discussed.

• **Incentive-Based Evaluation:** Soft-goals are modeled with the aid of SIG and are highly dependent on the expert knowledge of the software system and available metrics to measure the operationalized soft-goals. As with most techniques using models, the validity of the software artifacts, namely the goal models and accompanying utility functions, are threats to validity. Nonetheless, model-based techniques are increasingly used to handle system complexity at the design level.

• **Game-Theoretic Strategy Selection Engine:** An important underlying assumption of the proposed game-theoretic approach is the rationality of the SPS system and the attacker. However, attacker or the SPS system may not act rationally in real world mainly because of limited observations and available information. In a two-player zero-sum game, if a defender chooses a Nash equilibrium strategy, the rationality of the attacker is not important. We should take into account that any deviation from the Nash equilibrium decreases the cost of the defender while increases the the benefit of the attacker [4].

• **Modeling IBSP via Stochastic Multiplayer Game:** Another concern in the validity of the analysis is the number of parameters involved in the decision model that could affect the results, For example, the utility function can be defined in various ways. We have provided our rational for the defined utility function in Section 4.4.1, which we believe to be reasonable.

Despite of aforementioned threats to validity, the modeling and analysis via SMG are conducted aiming to understand the constraints and strength of the IBSP framework. IBSP framework can provide a rational guideline for addressing attack-type uncertainty using an incentive-based approach.

4.7 Summary

Most self-protection research to-date focuses on gathering information about the incoming attack and providing a defense strategy accordingly. However, an SPS system can benefit from fusing information about the attacker that has initiated the attack such as attackers' intention and hence taking quick and valid reactions to uncertain types of attacks. Motivated by the lack of sophisticated decision-making approaches that incorporate information from attackers in the context of self-protecting software systems, we have proposed a novel Incentive-Based Self-Protection (IBSP) framework. This framework allows for reasoning about defense strategies while considering (i) quality goals that align with the business goals of stakeholders such as performance, and (ii) malicious goals that motivate attacks on the system such as accessing sensitive data via an insider attack. The later consideration results in a decision-making engine that is not specific to a certain type of attack.

Furthermore, we map the decision model as a two-player game between the SPS system and the attacker in order to find the most proper countermeasure while considering the possible attack strategies. We argue that considering the intentions of attackers can produce more desirable decisions when dealing with attack type uncertainty due to the fusing information regarding the impact of attacker's intentions and strategies on the defense strategies. As a proof of concept, we present a formal modeling of our decision-model with the aid of probabilistic model checking of Stochastic Multiplayer Game (SMG) model [23] implemented in the probabilistic model-checker PRISM-games [24]. Our formal SMG modeling shows that IBSP framework is capable of making effective decisions.

This chapter has provided our vision on how to build the decision model that not only considers the quality goals of the system, but also acknowledges the malicious goals of attackers. We believe incentive-based self-protection can achieve breakthrough improvements in software security. Nevertheless, many challenges remain.

The next chapter investigates augmenting the decision-making engine to incorporate the success or failure of chosen countermeasures in the later countermeasure selections.

Chapter 5

Decision Making using Markov Games

Software security is defined by Gary McGraw [90] as: "The idea of engineering software so that it *continues to function correctly* under malicious attack". Improving techniques for intrusion *detection* and *prevention* has gained a lot of attention in the literature. On the other hand, intrusion *response* (attack mitigation techniques) are either static, or they are performed manually by security administrators after receiving an alert from the Intrusion Detection System (IDS) [137]. The delay between the intrusion *detection* and the intrusion *response* works to the advantage of attackers, helping them reach their intended malicious goals. To tackle this problem, Self-Protecting Software (SPS) systems are growing in importance. SPS systems are a class of autonomic systems capable of detecting and mitigating security threats at runtime [132].

The increasing number of successful application-layer attacks calls for new approaches for developing application-layer security. Most state-of-the-art SPS systems realize the deciding process with the aid of traditional rule-based [135], goal-based [99], or policy-based [7] approaches. These approaches either fail to protect a software system when it is facing attack-type uncertainty or become too complicated to capture and maintain all possible scenarios. We believe that today's well-planned attacks call for sophisticated countermeasure selection techniques that can outsmart such attacks.

SPS systems are a sub-class of Self-Adaptive Software (SAS) systems [71]. Uncertainty in making adaptation decisions in SAS is analyzed by adapting a possibilistic

method which is built on possibility theory to assess the positive and negative consequences of uncertainty [47]. Moreno et al. study decision-making under uncertainty for proactive self-adaptation [92]. Their approach addresses the uncertainty about the predictions of the future state of the environment in SAS systems. Incorporating attack-type uncertainty into the decision-making process is studied in the field of *network security*, e.g., wireless ad hoc networks [85], Mobile Ad hoc Networks (MANETs) [79], and network protection [53]. These approaches address attack-type uncertainty at the *network* layer using the network layer data, and hence they do not tackle such uncertainty in the *application-layer* attacks. In the field of *software security*, modeling and analyzing attack-type uncertainty in SPS systems is investigated in [46]. The proposed approach in [46] considers the uncertainty about the type of attack by incorporating the probability for each type of attack with the aid of Bayesian games.

In the Markov game technique adapted in [45], the application of game-theory in SPS systems demonstrates promising results in fusing the strategy of attackers. The cost and benefit of strategies and incorporating their impact on quality goals into modeling the interdependencies of strategies between the SPS system and the adversary is proposed in [44]. This approach is further extended to formulate the type of the adversary into the decision model of the Bayesian game decision-making technique in [46]. Markov game technique is applied for cyber security monitoring [28] and moving target defense [87]. What differentiates our proposed decision-making engine from other Markov-based techniques is the definition of the reward function and fusing the preferences of quality goals into the decision model.

In this chapter, we explore the possibility of learning the type of an attack with the aid of Markov game technique. Specifically, we propose a simple yet novel reward function that is employed by Markov game technique. The proposed learning-based approach aims at learning the best countermeasure against an uncertain attack-type. Learning is based on the reward/punishment received from the success/failure of applying the previous countermeasure. The proposed multi-objective reward function results in supporting multi-objective decision making. The objectives can be modeled and employed according to the information available to the software system and the stakeholders' preferred goals.

The rest of this chapter is organized as follows. Section 5.1 presents the notations used throughout this chapter. Section 5.2 provide an overview of the extended framework employing the Markov game technique. Section 5.3, 5.4, and 5.5 dive deeper into each phase of the extended framework: Modeling, Designing, and Realizing phases respectively. Section 5.6 exhibits the obtained results of the case study

that realizes the proposed Markov game decision-making engine. Finally, Section 5.7 summarizes the chapter.

5.1 Notations

Table 5.1 shows the notations used throughout this chapter. Sets are shown by calligraphic letters such as $\mathcal{X} = \{x_i\}$. Functions are illustrated by italic letters e.g., X. Matrices appear in bold capital letters e.g., $\mathbf{X} = [x(i,j)]$. Vectors are shown using bold small letters e.g., $\mathbf{x} = [x(i)]$. The optimal policy and the parameters are shown with Greek alphabet letters. Reward value and its magnitude is displayed using $\Re$ symbol.

5.2 A Markov Game Approach

To address the aforementioned problem, we employ Markov game technique in order to learn the type of an attack with the aid of the attack's impact on the quality goals (high-level objectives[1]) of an SPS system. The proposed approach learns the intention of the adversary as the defense mechanism responds to the attack and gets rewarded or punished based on the success or failure of the response action.

As discusses in our paper [45], we modified the well-known MAPE-K reference model [71] and illustrated the incorporation of Markov games into MAPE-K model. We showed how the raw monitored data can be filtered or transformed and subsequently analyzed to be employed in the Markov game formulation. In this chapter, we provide a more systematic approach on how to engineer the adaptation manager. More specifically: (i) we provide steps for modeling high-level quality goals to low-level measurable goals that can be quantified with system/user attributes, (ii) we formalize the state of the system and the reward function calculation based on the satisfaction of the defined quality goals, and (iii) we evaluate the proposed approach and the affect of its parameters in more detail.

Before giving the theoretical design and implementation details, we provide a high-level architectural view of the proposed *MARkov Game decIsion-making engiNe (MARGIN)* illustrated in Fig. 5.1. To build an SPS system from a software

[1]Objective and goal are different in some contexts. For instance, Keeney et al. use objectives in a higher level of abstraction [70]. In this article, without loss of generality, we assume that goals and objectives are the same. Therefore, we use them interchangeably hereafter.

Table 5.1: Summary of Notations Used

Notation	Definition
$\mathcal{A} = \{a_i\}$	Set of countermeasures/response actions
$\mathcal{AT} = \{at_i\}$	Set of attributes
$\mathcal{G} = \{g_i\}$	Set of quality goals
$\mathcal{O} = \{o_i\}$	Set of possible attacks
$\mathcal{S} = \{s_i\}$	Set of system states
$Cost(a_i, o_j)$	The cost fucntion for the response action $a \in \mathcal{A}$ against action $o \in \mathcal{O}$
$Des(s_i)$	The desirability of a state of the system
$R(s_i, a_k, o_m, s_j)$	The reward function for moving from state s_i to s_j via countermeasure a_k and attacker's action o_m
$Sat(g_i, s_j)$	The satisfaction of a quality goal g_i at state s_j
$T(s_i, a_k, o_m, s_j)$	The state transition function for moving from state s_i to s_j via countermeasure a_k and attacker's action o_m
$U(g_i, at_j, s_k)$	The utility value of the g_i goal and attribute at_j at state s_k
$\mathbf{p} = [p(g_i)]$	$p(g_i)$ is the priority of quality goal g_i from stakeholders' opinion
$\mathbf{v} = [v(s_i)]$	$v(s_i)$ is the expected reward for the optimal policy at state s_i
$\mathbf{GT} = [gt(s_i, a_j, o_k)]$	$gt(s_i, a_j, o_k)$ is the quality of action a_j against action o_k in state s_i
$\mathbf{PD} = [pd(s_i, a_j)]$	$pd(s_i, a_j)$ is the discrete probability distribution for action a_j in state s_i
$\mathbf{W} = [w(g_i, at_j)]$	$w(g_i, at_j)$ is the weight of attribute at_j's contribution in quality goal g_i
$PD(\mathcal{A})$	The probability distribution over set $\mathcal{A}$
π	The optimal policy
α	The learning rate
γ	The discount factor
ϵ	The explore rate
$\Re$	The reward value
$\Re^+/\Re^-$	Positive/negative magnitude of the reward value

system, we introduce three high-level components for implementing the adaptive loop, namely: (i) the *IDS component* is responsible for monitoring and analyzing the incoming traffic of requests, (ii) the *GAAM component* formulates the high-level quality goals of the system with the aid of Goal-Action-Attribute Model (GAAM) [110], and (iii) the *MARGIN component*, which is the focal point of this chapter, is respon-

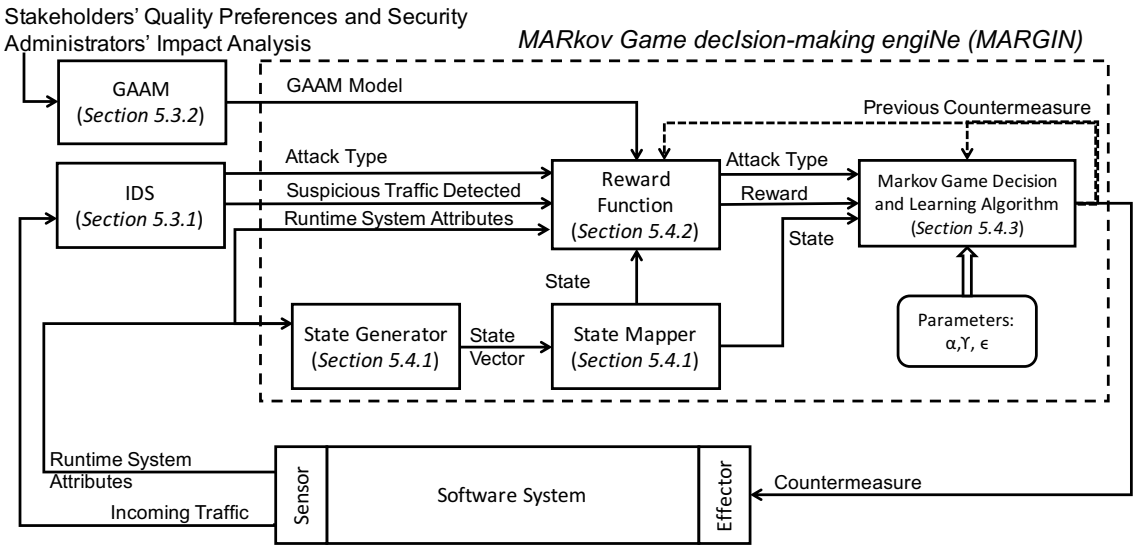


Figure 5.1: High-level Architecture of MARGIN in a SPS System

sible for choosing the next countermeasure (adaptation action).

The *IDS component* can be integrated with the aid of existing off-the-shelf IDSs. Building the *GAAM component* requires the involvement of stakeholders and security administrators to prioritize the impact of countermeasures and attacks on both security and non-security goals of the SPS system. For instance, the confidentiality of sensitive information in a web application of a financial institute is considered as a high priority goal but it can be ignored in a news website which does not store confidential information about its users. Consequently, the GAAM needs to be modeled in a way that captures the preferences of quality goals. Building the *MARGIN component* consists of assembling four sub-components (depicted in Fig. 5.1):

- **State Generator**: This component gets the value of the monitored runtime attributes and discretizes these values according to predefined thresholds that are carefully chosen.
- **State Mapper**: This component takes the discretized values from the *State Generator* component and assigns a single number that represents the whole vector of attribute values. This number serves as the state of the SPS system.

- **Reward Function**: This component systematically calculates the success or failure of the previous countermeasures taken against an attack based on their impacts on the quality goals defined in GAAM model. The reward value reflects the interdependency of a countermeasure's effectiveness on how the system is attacked and the interdependency of an attack's success on how the system is protected (such interdependency is neglected in the majority of SPS systems [84]).
- **Markov Game Decision and Learning Algorithm**: This component is responsible for: (i) updating the payoff tables, and (ii) selecting the next proper countermeasure.

In MARGIN, the objective is to automatically select the most proper countermeasure, using a decision model which: (i) provides learning-based dynamism of adversary's type of attack by considering the response actions taken so far, and (ii) supports multi-objective decision making by incorporating priorities and preferences of the SPS system's quality goals. We next explain how each objective is met.

5.3 Modeling: Intrusion Detection and Quality Goals Model

For the rest of this chapter, we take the motivating scenario, described in Section 5.1 as a guideline to define a two-player game. Before detailing the game, we need to detect the intrusion and formalize the quality (security and non-security) goals as detailed in the next two subsections.

5.3.1 The Intrusion Detection System

This component has two main responsibilities: (i) detecting suspicious traffic and sending an alarm to the decision-making engine, and (ii) gathering and analyzing malicious symptoms in order to identify the type of intrusion, then sending the detected attack-type to the decision-making engine. Since such analysis takes time, the SPS system provides the response action as soon as the malicious traffic is detected even though the actual type of the attack is still ambiguous to the IDS. When the

type of an attack is recognized by gathering enough information, this information is send to the decision-making engine to be used in the learning process.

Our current implementation of the IDS uses simplistic solutions to determine the likelihood that a request is malicious. However, existing off-the-shelf IDSs can also be used. Techniques such as user-behavior analysis [129] can also be used to determine whether a client is malicious. Determining this information is a challenge in its own right, and not the focus of this research.

5.3.2 The Goal-Action-Attribute Model

This component provides a formal way to describe the SPS system's security and non-security goals, and the impact of possible response actions (countermeasures) on each goal using runtime system attributes. The ability of the GAAM to model system goals, actions, and monitored attributes along with their dependencies makes it a suitable model for describing the impact of countermeasures on the high-level quality goals.

Building the GAAM is the only step that is performed offline by collecting information from stakeholders and security administrators. A GAAM's structure is expressed in a node hierarchy, allowing one to compose and abstract software goals such as *availability* into a number of measurable properties that facilitate tracing goals. For example, the number of requests in a queue to reach the server can be used to evaluate the goal of achieving the possible maximum service availability. If the number of requests is higher than a certain threshold, the server will not be able to serve all the requests, and some of the requests will be timed-out. Hence, the server is not available to those requests.

Suppose we have a set of quality goals. These goals represent the objectives of the SPS system. More formally, quality goals are represented as a set of n variables $\mathcal{G} = \{g_1...g_n\}$. In addition, we measure m attributes from the SPS system, such as load, throughput, response time, etc. These attributes are represented in a set $\mathcal{AT} = \{at_1, ..., at_m\}$ GAAM uses a matrix called the activation matrix ($\mathbf{W} = [w(g_i, at_j)], i = 1..n, j = 1..m$), defined by security administrators to represent the relationship among n quality goals and m system attributes. The values in the **AM** matrix show how much each attribute participates in satisfying a goal.

Additionally, each goal has certain priority over other goals. Priorities come from stakeholders' opinions and are represented by a priority vector $\mathbf{p} = [p(g_i)]$ with values

in [0..1]. We assume that $\sum_{i=1}^{n} p(g_i) = 1$. Finally, a set of possible response actions $\mathcal{A}$ and a set of possible attacks $\mathcal{O}$ are recognized.

The output of the GAAM component is a systematic model that can be used to calculate reward values in the *Reward Function* component. It is noteworthy that the rationale behind using the GAAM in MARGIN, rather than having the stakeholders and security administrators to directly model the priorities and preferences in MARGIN, is that GAAM models are easier to understand and to design manually (partially because of their tree structure). The purpose of the GAAM component is to allow stakeholders and security administrators to define high-level security properties through easy-to-understand linguistic terms.

5.4 Designing: Markov Game Decision-Making Engine

In this section, we elaborate on the components that build the proposed decision-making engine.

5.4.1 The State Generator and the State Mapper

Combining different forms of evidence leads to more-precise attack detection, especially evidence from the impact of an attack on the software system. In an SPS system, possible evidence is gathered from monitored attributes and translated into a *state* of the system. Runtime attributes that indicate the level of satisfactory of the defined goals in an SPS system are used to represent the state space in the SPS system. Values of these attributes can be discrete or continuous. Discrete attributes, e.g. the load of the system, can take certain values (e.g. "high", "normal", and "low"). Continuous attributes indicate a measurement at a certain point. For example, the number of request a user makes per hour.

Attributes can have continuous domains, sometimes leading to the problem of state explosion. To deal with this problem, the *State Generator* component discretizes attribute values according to predefined thresholds. Each possible tuple of attribute values $(at_1, ..., at_m)$ is defined as a state of the system. The *State Mapper* takes the vector of discretized values and maps them to a single number that represents the whole vector. This number serves as the state of the SPS system.

5.4.2 The Reward Function

The reward function is one of the most-important factors in formulating the MARGIN. It is what guides the decision-making engine to learn from the success or failure of previous countermeasures. In [45], we study two types of reward functions that are defined using high level rule-based definitions. Such reward functions are highly dependent on the knowledge of the administrator that defines the rules and thresholds. In this chapter, in order to make the reward function definition independent of such expert knowledge, we formalize the reward value based on the effectiveness of applying a countermeasure which can be reflected by its impact on the quality goals of the system. In our proposed reward function, the interdependency between the defense strategy and the attack strategy is captured in the reward function by measuring the achievement of quality goals.

In other words, the reward function represents the preferences over countermeasures by considering the cost and benefit of each countermeasure, using their effect on quality goals. Incorporating such information into modeling the interdependencies of strategies between the SPS system and the attacker is proposed in [44]. In this section, we formulate this information into the reward function in order to learn how favorable a countermeasure is in face of an ambiguous attack-type. In addition, incorporating high-level objectives of the system into the decision model results in supporting multi-objective decision making.

To measure the satisfaction of quality goals, we characterize attribute values using *utility functions*. Attribute values can be either: (i) continuous, or (ii) discrete. For instance, "response time" attributes can have continuous values stated in millisecond (ms) units, whereas "user annoyance" can be reflected in discrete values such as *annoyed* and *satisfied*. In both cases, utility functions are defined by an explicit set of value pairs that map attribute values to utility values between 0 and 1. In case of continuous attributes, intermediate points are linearly interpolated. Such utility function definition is adopted from the work by Schmerl et al. in [111]. Table 5.2 provides examples of utility functions for the "Usability" goal, which is measured by one continuous attribute ("response time") and one discrete attribute ("user annoyance"). The utility function $U_{Usability,\,ResponseTime}$ maps low response time with maximum utility, while response times higher than 2000 ms are highly penalized with low utility. For response times above 5000 ms, the utility is considered to be 0. The utility function $U_{Usability,\,User\,Annoyance}$ maps user satisfaction to the high utility value of 1, whereas "user annoyance" caused by issuing puzzle tests yields the low utility value of 0.

Table 5.2: Utility Function Examples for the "*Usability*" Goal and Its Attributes: "*Response Time*" and "*User Annoyance*" at the state of "*Running*"

$U(Usability, Response\,Time, Running)$	$U(Usability, User\,Annoyance, Running)$
0 : 1.00 1000 : 0.50	satisfied : 1.00
100 : 1.00 2000 : 0.25	annoyed : 0.00
500 : 0.75 5000 : 0.00	

Each quality goal is affected by the value of one or more attributes. The matrix **AM** in the GAAM represents the impact of attributes on quality goals, using weights $(w(g_i, at_j))$. Without loss of generality, we assume $\sum_{i=1}^{n} w(g_i, at_j) = 1$, and each weight $w(g_i, at_j)$ has a value between 0 and 1. Hence, the satisfaction of a quality goal g_i at state $s \in \mathcal{S}$ is defined as:

$$Sat(g_i, s_k) = \sum_{j=1}^{m} w(g_i, at_j) * U(g_i, at_j, s_k), \tag{5.1}$$

where $w(g_i, at_j)$ is the the effect of the j_{th} attribute on the g_i and $U(g_i, at_j, s_k)$ is the utility value of the g_i goal and attribute at_j at state $s_k \in \mathcal{S}$. Priorities and preferences of quality goals are captured by assigning a specific weight $\mathbf{p} = \{p(g_1)...p(g_n)\}$ to each. We measure how "desirable" a state is according to the preferences of the defined goals:

$$Des(s_k) = \sum_{i=1}^{n} p(g_i) * Sat(g_i, s_k). \tag{5.2}$$

In addition to considering the quality goals of the system, we must also consider stakeholders, whose goal is to minimize operational costs. The latter objective is reflected in the reward function by incorporating the operational cost. $Cost(a_i, o_j)$ is the cost function for the response action $a_i \in \mathcal{A}$ against action $o_j \in \mathcal{O}$, regardless of the source and destination states. Note that the SPS system can chose not to take any action; in which case, no cost is assigned. Now we can define the reward function as represented in Equation 5.3 as follows:

$$R(s_i, a_k, o_m, s_j) = Des(s_j) - Des(s_i) - Cost(a_k, o_m). \tag{5.3}$$

In Equation 5.3, $R(s_i, a_k, o_m, s_j)$ is the reward gained by moving from state $s_i \in \mathcal{S}$ to state $s_j \in \mathcal{S}$ via the response action $a_k \in \mathcal{A}$ and the opponent's (attacker's) action

$o_m \in \mathcal{O}$. The reward value can be either positive or negative represented by $\Re^+/\Re^-$. Next, we show how the defined reward function is used in the Markov game decision and learning algorithm.

5.4.3 The Markov Game Decision and Learning Algorithm

The success or failure of an attack is highly dependent on how the system is protected, and similarly, the effectiveness of a defense mechanism is dependent on the type of attack [84]. In such situations, a natural fit is to formulate the decision making as a game between the SPS system and the attacker. Markov games, also known as stochastic games, are the extension of the single agent Markov Decision Process (MDP) to the multi-agent case [82]. This makes the Markov game formulation a powerful tool for modeling decision-making in dynamic multi-agent environments. In our game model, we plan to incorporate the feedback from previous actions taken by the system. Markov game technique incorporates the reward from the previous action. Hence, this technique is employed to solve our modeled two-player game. In our proposed approach, Markov game modeling allows us (i) to incorporate the attacker's actions in the decision model, and (ii) to reason stochastically about uncertainty in the attack-type.

Definition 1. (Two-Player Markov Game) A Markov game, sometimes called a stochastic game, is defined by a set of states $\mathcal{S}$; a collection of action sets $\mathcal{A}, \mathcal{O}$, one for each agent (player) in the environment; and a probabilistic transition function $T(s_i, a_k, o_m, s_j)$ that controls state transitions by the current state and one action from each agent. Moreover, each agent has an associated reward function, $R(s_i, a_k, o_m, s_j) = \Re$, where $\Re$ is the set of real numbers. Each agent attempts to maximize its expected sum of discounted rewards. A *discount factor*, $0 \leq \gamma < 1$ controls how much effect future rewards have on the optimal decisions. Small values of γ emphasize near-term gain and larger values give significant weight to later rewards.

We model the interactions between the attacker and the decision-making engine in MARGIN as a two-player zero-sum Markov game [82] in which the number of players is two and they have diametrically opposed goals. This opposition allows using a single reward function that one player tries to maximize and the other tries to minimize. Using Markov games enables us to reason about the behavior of players using strategies. A *strategy* selects which action to take in every state.

Definition 2. (Strategy) The *strategy* of each player in Markov games, also referred to as its *policy*, is a function that maps states to discrete probability distributions over actions of that player. π is the optimal policy that maximizes the expected sum of discounted reward.

Thus, actions are sampled according to their probability distribution. A strategy is said to be a *pure-strategy* if probability distribution over set $\mathcal{A}$ assigns a probability 1 to a certain action; otherwise, it is said to be a *mixed-strategy*. In Markov games, probabilistic action choices are needed because of (i) the player's uncertainty about its opponent's current action, and (ii) the player's need to prevent opponents guessing its strategy [82]. Hence, the strategy of the SPS system in our proposed approach is a probability distribution over the response actions (countermeasures). The optimal policy in Markov games is defined as the strategy that maximizes the reward in the worst case. This resolution (i) eliminates the choice of the opponent, and (ii) evaluates each policy with respect to the opponent that makes it look the worst [82].

In this chapter, as defined in Section 5.3.2, $\mathcal{A}$ is the set of response actions (countermeasures) offered by the SPS system, and $\mathcal{O}$ is an adversary's set of actions. In our defined two-player zero-sum Markov game, the value of each state $v(s_i)$ is defined as the expected reward for the optimal policy, starting from state s_i:

$$v(s_i) \leftarrow \max_{\pi \in PD(\mathcal{A})} \min_{o_k \in \mathcal{O}} \sum_{a_j \in \mathcal{A}} gt(s_i, a_j, o_k) * \pi. \tag{5.4}$$

The payoff table **GT** represents the quality of action a_k against action o_m in state s_i. The payoffs in **GT** are calculated as in Equation 5.5, in which $\Re$ is the reward when making a transition from s_i to s_j after taking action a_k against action o_m ($\Re = R(s_i, a_k, o_m, s_j)$). Note that the value of r is calculated with the aid of the reward function defined in the previous section. The discount factor γ, as mentioned above, determines the importance of earlier versus later rewards.

$$gt(s_i, a_k, o_m) \leftarrow \Re + \gamma \sum_{s_j \in \mathcal{S}} T(s_i, a_k, o_m, s_j) * v(s_j). \tag{5.5}$$

In many Markov game models such as ours, the state transition function $T(s_i, a_k, o_m, s_j)$ is not known in advance. Accordingly, an alternative approach involves performing updates without the use of the transition function:

$$gt(s_i, a_k, o_m) \leftarrow (1-\alpha) * gt(s_i, a_k, o_m) + \alpha * (\Re + \gamma * v(s_j)). \tag{5.6}$$

In Equation 5.6, α is the *learning rate* which determines to what extent the newly acquired information will override the old information. This learning technique that updates the **GT** payoff tables is called *minimax-Q* [82]. The decision and learning model in MARGIN is defined using the minimax-Q algorithm in Markov games. Minimax-Q is similar to *Q-learning* algorithm [118] for finding optimal policies with the *minimax* function replacing the *max* function in calculating the value of a state.

The learning rate α in the original minimax-Q algorithm proposed in [82] decreases over time by a factor of *decay*, which suggests that the learning phase stops after a certain time. However, in the dynamic environment that we are modeling, it is hard to predict when an attack is going to occur and how long the learning frame should last to guarantee that enough knowledge is acquired. Furthermore, the strategy of an attacker can change over time. For these reasons, in our decision model, the learning rate does not decay.

The Markov game decision and learning algorithm that is implemented in MARGIN is shown in Algorithm 2. The algorithm contains three main parts: *initialization*, *learning*, and *action selection*, as explained below.

• **Initialization**: In Algorithm 1, lines 1-9 initialize the variables. Table **GT** contains the payoff of each countermeasure against possible attacks in each state of the system. The payoff table **GT** can be either initialized to 0 as shown in lines 1-3, or according to the expectation of a system admin. The payoff table gets updated as the system continues to run. In Markov game, each state has a value represented by $v(s_i)$. Lines 4-6 initialize state values to their default value. The probability distribution of the countermeasures in each state ($pd(s_i, a_k)$) are initialized to ($1/number\ of\ countermeasures$) as the selection probabilities of all the countermeasures are equal initially (lines 7-9).

• **Learning**: Learning happens after the algorithm receives reward $\Re$ for moving from state s' to s'' via action a' and the opponent's action o' (all are provided as inputs to the algorithm). Lines 10-12 provide the learning technique in MARGIN using the reward value calculated by the reward function in Section 5.4.2. After a certain time, the type of the current attack is detected by the IDS, and the SPS system moves from state s' to s'' via action a' and opponent's action o'. The reward value is calculated using Equation 5.3 in Section 5.4.2 and the payoff table **GT** is updated accordingly (line 10). In line 12, the optimal policy is calculated using linear programming, by finding the optimal discrete probability distribution over the set of response actions $\mathcal{A}$

Algorithm 2: Markov Game Decision and Learning Algorithm

Input : $a' \in \mathcal{A}$ - the previous countermeasure
Input : $o' \in \mathcal{O}$ - the previous attack
Input : $s' \in \mathcal{S}$ - the previous state
Input : $s'' \in \mathcal{S}$ - the current state
Input : $\Re$ - the reward value computed using the reward function
Output : $a'' \in \mathcal{A}$ - the chosen countermeasure to be triggered
Parameter: α - the learning rate
Parameter: γ - the discount factor
Parameter: ϵ - the explore rate

`// Initialization`

forall $s_i \in \mathcal{S}, a_k \in \mathcal{A}, and\ o_m \in \mathcal{O}$ **do**
    $\mathbf{GT}(s_i, a_k, o_m) \leftarrow 0_{|\mathcal{S}|\times|\mathcal{A}|\times|\mathcal{O}|}$
end
forall $s_i \in \mathcal{S}$ **do**
    $v(s_i) \leftarrow 1_{|\mathcal{S}|}$
end
forall $s_i \in \mathcal{S}\ and\ a_k \in \mathcal{A}$ **do**
    $pd(s_i, a_k) \leftarrow 1/|\mathcal{A}|_{|\mathcal{S}|\times|\mathcal{A}|}$
end

`// Learning`

$gt(s', a', o') \leftarrow (1-\alpha) * gt(s', a', o') + \alpha * (\Re + \gamma * v(s'))$
Use linear programming to find optimal policy over action set A in previous state s' such that: $pd(s', a_k) \leftarrow \underset{pd(s',a_k)}{\text{argmax}} \min_{o_m \in \mathcal{O}} \sum_{a_k \in \mathcal{A}} gt(s', a_k, o_m) * pd(s', a_k)$
$v(s') \leftarrow \max_{\pi \in PD(\mathcal{A})} \min_{o_m \in \mathcal{O}} \sum_{a_k \in \mathcal{A}} gt(s', a_k, o_m) * \pi$

`// Action Selection`

r = random()
if $r < \epsilon$ **then**
    $a'' \leftarrow$ Choose a random action from $\mathcal{A}$
else
    $a'' \leftarrow$ Choose an action from $\mathcal{A}$ according to the probability distributions in **PD** and the current state s''
end
return a''

in state $s' \in \mathcal{S}$. The probability distribution $pd(s', a_k)$ is used in the algorithm to select the next-best response action based on the current state of the system. The value of each state $v(s')$ is defined as the expected reward for the optimal policy in state s. Line 12 updates $v(s')$ according to the optimal policy $pd(s', a_k)$ calculated in the previous line. The learning process results in updating the payoff tables based on the success or failure of a countermeasure. Hence, the values in the payoff table **GT** represent the impact of countermeasures on the quality goals against different types of attacks. Updating the payoff table guides future action selections toward the right direction: choosing a countermeasure that maximizes the minimum possible payoff.

• **Action Selection**: Lines 13-18 present the action selection process. Action selection is either based on exploring the deviation from the current policy using the ϵ probability or based on the current probability distribution values. In line 15, with probability of ϵ, the algorithm returns an action uniformly at random. Otherwise, based on the current state s'', it returns an action $a'' \in \mathcal{A}$ with the probability of $pd(s'', a'')$ (line 17). At the time that MARGIN chooses the next action, the opponent action (the type of attack) is unknown to MARGIN, so action selection involves finding an optimal policy that minimizes the maximum damage to the system, which is the essence of using the *minimax* technique. The MARGIN acts conservatively by considering the worst-case scenario in finding the optimal policy.

As mentioned earlier, there is a need for specific decision-making approaches in SPS systems to respond to attacks as soon as malicious activity is detected, even before the type of attack is clear to the defense system. Such capability is traditionally ignored by static policy-based/rule-based approaches. In our proposed approach, to cope with uncertainties about the attacker's possible type of attack, we incorporate the impact of countermeasures that are taken in each attack-type uncertainty situation, in order to learn the proper (more conservative) countermeasure. Moreover, using the *minimax* technique, the next possible action of the rational attacker which maximizes the damage to the system is taken into account when selecting the response.

5.5 Realizing: Case Study of a Web Application Using Simulink

This section describes experimental results to demonstrate the value of the proposed decision-making engine when the type of the ongoing attack has not yet been detected by the IDS. For this purpose, we model and simulate scenarios in which the type of

attack is unknown at the time of making the adaptation decision. The implemented scenarios are general enough to be used for different types of application-layer attacks (e.g., an insider attack), but we cannot claim that the following experiments evaluate MARGIN for all possible attack types. The focus of our experiments is to evaluate the efficiency of MARGIN in learning the proper countermeasure selection by the time the type of the attack is detected by the IDS. Accordingly, the objectives of our experiments are to answer the following research questions:

- **RQ2.1:** Can MARGIN learn to select a proper countermeasure?
- **RQ2.2:** What is the effect of the *cost* of a countermeasure in the action selection?
- **RQ2.3:** What is the impact of the *explore rate* on the performance of the action selection?
- **RQ2.4:** What is the effect of the *discount factor* on the performance of the action selection?
- **RQ2.5:** What is the effect of the *learning rate* on the performance of the action selection?
- **RQ2.6:** Is learning a good idea? (How does the proposed technique perform comparing with other techniques?)

In the rest of this chapter, we describe the implemented attack scenarios and the experimental setup. The detail for implementation of the three main components (IDS, GAAM, and MARGIN) is provided, followed by a review of the results and threats to validity.

5.5.1 Attack-Type Uncertainty Scenarios

For the experiments, we define various possible scenarios in case of attack-type uncertainty. These scenarios have no special assumptions about the attackers and the architecture of the system. In the defined scenarios the SPS system can be targeted by two major threats that exhibit very similar behavior in the beginning but have different malicious intentions (as described in Section 5.1). The first attack-type is an application-layer DoS attack [20] that targets the *availability* of the system, and the second one is an insider attack [21] that threatens the *confidentiality* of the system.

Both of these attack-types require the attacker to send requests to download a set of files in order to achieve their malicious goals. Thus, the attacker downloads files either (i) to overload the server, or (ii) to retrieve as much confidential information as possible.

In the defined attack scenarios, DoS attacker and insider attacker aim at staying hidden from the IDS. Hence, they both send requests to the system similar to the requests from a regular user. Hence, the set of the adversary's actions, $\mathcal{O}$, can be defined as: {*DoS Attack*, *Insider Attack*, and *No Attack*}. In the case of a *DoS Attack*, the best response action is to identify the user which is sending malicious requests, by issuing a puzzle test such as a CAPTCHA test [126] to the sender. If the user fails the test, then the requests are initiated from *DoS botnets* [89] and are considered to be a malicious DoS attack. A botnet is a network of compromised machines (bots) under the control of an attacker (the botnet herder) [57].

In the case of an *Insider Attack*, the attacker can successfully pass the puzzle challenge, so the best response is to *Drop* the malicious request. It is also possible that the suspicious user is not an attacker and has no malicious intention (*No Attack*). For example, a regular user is downloading a large number of files. In such a case, the SPS system does not need to trigger a countermeasure (*No Defense*). Consequently, the response action set, $\mathcal{A}$, for these attack types respectively are defined as: {*Issue Puzzle*, *Drop*, and *No Defense*}.

Considering the two defined threats to the experimental SPS system, the implemented MARGIN is evaluated for the following attack-type uncertainty scenarios:

- **Scenario 1:** In the first scenario, the SPS system is under a DoS attack. However, both DoS attacker and regular users send similar requests to the system. The DoS attacker has the malicious intention to decrease the availability of the system whereas the regular user has no malicious intention.

- **Scenario 2:** In the second scenario, the SPS system is under an insider attack which have different impacts on quality goals compared to a DoS attack. Comparable to the previous scenario, both insider attacker and regular users send similar requests to the system whereas their requests differ in their intentions.

- **Scenario 3:** In the third scenario, the SPS system is targeted by both DoS and insider attacks simultaneously. In this scenario, the defense mechanism need to chose the proper action according to DoS attackers, insider attackers, and regular users. This scenario aims at challenging how the proposed approach copes with multiple simultaneous attacks.

- **Scenario 4:** In the forth scenario, the SPS system is under a DoS attack first, followed by an insider attack. Both of these attacks are initiated by the same attacker. Hence, this scenario challenges the learning ability of the proposed decision model when the type of attack changes dynamically.

In each scenarios, the attack/attacks occur multiple times and for each occurrence, the response action selected by the MARGIN is reported.

5.5.2 Experiment Setup

As a proof of concept, a web application is simulated, and has been chosen because such applications are the primary targets for attacks in which attackers mimic the behavior of legitimate users to stay hidden from intrusion detection systems [127]. The simulation is implemented using the SimEvents toolbox of MATLAB Simulink [66]. SimEvents provides a Discrete-Event Simulation (DES) model of computation, which is a proper fit for a web-application simulation. In an event-based simulation, the entire duration of the simulation study is a sequence of events.

The simulated SPS system[2] is a web application based on a queue-server model supplied with sensors and effectors. The sensors monitor the system and track user behavior. The effectors execute the adaptable action during runtime. The main server in the web application handles requests such as HTTP-GET requests and is equipped with a FIFO queue that stores the incoming requests and dispatches them to the server as soon as the server becomes available.

We simulate application-layer DoS attacks, insider attacks, and regular user requests by generating traffic on the system. The sending rate of requests coming from legitimate users is similar to that of malicious users. The attacks are simulated through a scaled simulation of an insider attack in a chemical company [21]. The scaled simulation is also used by Bailey et al. [7] to evaluate their insider attack mitigation approach. In this scaled simulation, the attacker's number of downloads is 15 times greater than a normal user's [21]. This number of downloads can also indicate an application-layer DoS attack or an active user with no malicious intention temporary downloading a higher-than-normal number of files. Hence, identifying the type of attack requires more symptom analysis by the IDS. In the following, we provide details of each component implemented in the simulated model.

[2]An implementation of MARGIN technique is available at https://github.com/mahsa-emamitaba/MARGIN-Simulink

5.5.3 MARGIN Realization

The decision-making engine in the simulated model develops the MARGIN described in Section 5.2. The inputs to the MARGIN are the GAAM model, the attributes that indicate the state of the system, and the type of the previous intrusion detected by the IDS. The output of the MARGIN is a response action to mitigate the current intrusion.

The Intrusion Detection System

The applicability of tackling attack-type uncertainty with a game-theoretic approach is evaluated by generating traffic on the server. The IDS recognizes suspicious clients from benign clients by monitoring the number of received requests. However, distinguishing the type of attack, requires more analysis of various symptoms of the attack, for example, its effect on the server's load or the number of requests to access sensitive information. Next, we model the goals, actions, and attributes used in the implemented experiments with the aid of the GAAM modeling technique.

The Goal-Action-Attribute Model

The specification of GAAM consists of recognizing the goals, attributes, and response actions in the system. Fig. 5.2 schematically depicts relationships between entities. Beside the figure, the list of actions and attributes are available. Each goal here is reflected by the value of one or two attributes. Hence, the weight w_{22} and w_{23} for attributes of g_2 in Equation 5.1 is set to 1/2. The preferences over four of the five leaf goals are the same in our example. The value of $p_1 ... p_4$ in Equation 5.2 is set to 1/6. The last goal, which is related to the confidentiality goal is given a higher preference of 1/3. The set of attributes defined here is further used to define the states of the SPS system. The values of these attributes are discretized to avoid state space explosion.

The State Generator and the State Mapper

In the implemented MARGIN, the *State Generator*, *State Mapper* are implemented in MATLAB functions. The former function discretizes the attribute values and the latter function maps the discretized values to a single number.

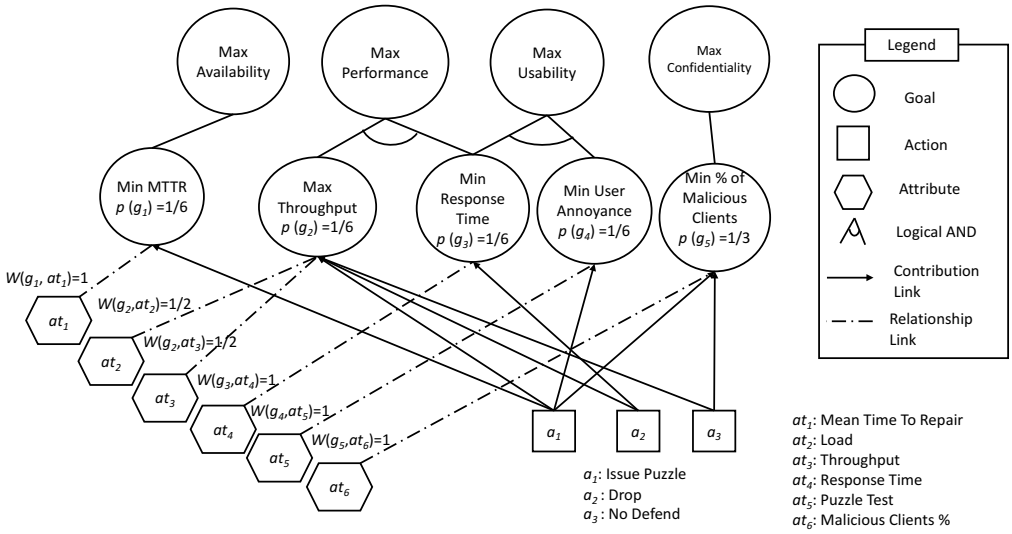


Figure 5.2: Composing GAAM for the Experimental Evaluation

The Reward Function

In the implemented MARGIN, the *Reward Function* is implemented as a MATLAB function. The priorities determined in the modeled GAAM in Fig. 5.2 are incorporated as defined in Equations 5.1-5.3.

The Markov Game Decision and Learning Algorithm

The Markov game algorithm, which is discussed in Section 4.5 in detail, is implemented using MATLAB functions. In Markov games, the payoff table **GT** represents the countermeasure preferences over each type of attack. Table 5.3 shows an example of the convergence of an updated **GT** table in our simulation model with the aid of the magnitude of reward values for each cell. The payoffs in the **GT** table represent the interdependency of countermeasures and attacks. In the proposed decision model, we define rewards of applying each countermeasure based on denial or satisfaction of the defined quality goals. The reward aids in considering the interdependency of the applied countermeasure with the type of attack.

Table 5.3: A **GT** Table Representing Interdependencies of Countermeasures and Attacks

Action	**DoS Attack**	**Insider Attack**	**No Attack**
Issue Puzzle	$\Re^{++}$	$\Re^{--}$	$\Re^{-}$
Drop	$\Re^{+}$	$\Re^{++}$	$\Re^{-}$
No Defense	$\Re^{-}$	$\Re^{-}$	0

5.6 Obtained Results

In the experimental evaluation, the goal is to observe whether the decision model learns which countermeasure works best against an uncertain attack-type. Before describing the results, we give an overview of the measures used for evaluating the quality of a decision making technique in response to a request from a regular user or an attacker.

A request to the SPS system can be handled in one of the following ways:

- **Successfully handled** – When the system provides the desired response to the request.
- **Rejected** – When the system either drops the request or issues a puzzle challenge and the requester fails to solve the puzzle.
- **Timed out** – When the system fails to provide a response to the request within a certain time.

It should be noted that we provide experimental results for a time period during which the type and maliciousness of the incoming requests are unknown to the IDS. When a request, initiated by an attacker is handled successfully by the SPS system, it causes degradation of the SPS system's quality goals' satisfaction. Hence, the lower the number of successful attack requests, the better. In such cases, the best response is to reject such malicious requests by either dropping them or issuing a puzzle.

Attack requests such as DoS attack requests or high-rate insider attack requests saturate system resources and cause the SPS system to fail in responding to regular users' requests. This results in those non-malicious requests getting timed out. In

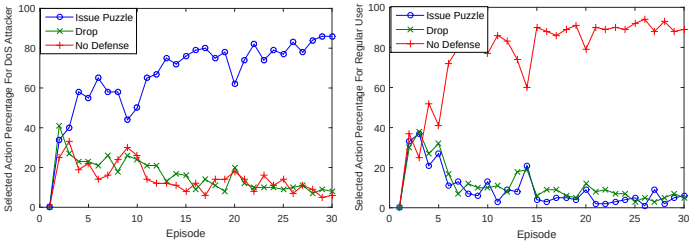
addition, while the MARGIN learns the proper countermeasure, the SPS system may reject (drop/issue challenge) some requests from regular users. Getting no response as a result of timeout or getting rejected by the SPS system affects these regular users' experience. Hence, in our experiments, we evaluate to what extent regular users are affected by the selected countermeasure. In our evaluation, the higher the number of successful requests for regular users, the better.

Given the research questions stated earlier, we have performed six sets of experiments to evaluate MARGIN. In each set of experiments, we explore the four attack scenarios introduced in Section 5.5.1. For each scenario, we run the experiment 100 times; Each run includes 30 actions by each of the attackers and 30 actions by a regular user. In other words, for each user, the MARGIN can choose 30 actions in response to 30 incoming requests. Each of the actions are selected in one episode of the simulation. Hence, each run includes 30 episodes. For each user (DoS attacker, insider attacker, regular user, mix attacker), the percentage values of *Success*, *Reject*, and *Timeout* requests out of 100 runs are reported. The following subsections provide the relevant answers for each of the research questions, followed by the threats to validity of the conducted experiment.

5.6.1 RQ2.1: Can MARGIN learn to select a proper countermeasure?

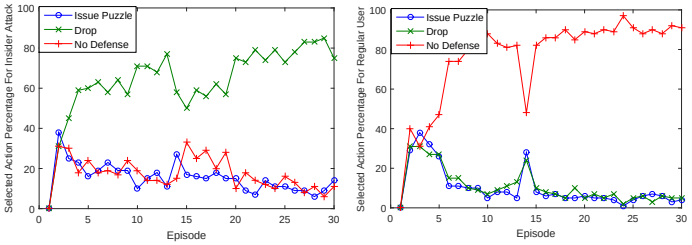
In this set of experiments, the minimax-Q parameters are set as follows: the discount factor, γ, is set to 0.5, and the learning rate, α, is set to 0.5. As for the ϵ, we set the value to 0.1 so that the system acts more conservatively. Figs 5.3a, 5.3b, 5.3c, and 5.3d depict the percentage of countermeasure selection for 100 executions over time for scenario 1, 2, 3, and 4 respectively. In each execution, MARGIN selects a response action 30 times (70 times in scenario 4).

As seen in the figures, MARGIN is able to learn the proper countermeasure after a few episodes. It continues to learn although explores other countermeasures randomly as well. In both cases (i) of Fig. 5.3a and Fig. 5.3b, MARGIN learns the proper countermeasures against the attacker. In the former, the proper countermeasure against the DoS attack is to issue puzzle and in the latter, the proper countermeasure agains the insider attack is to drop the request. In case (ii) of Fig. 5.3a and Fig. 5.3b, the requests are coming from a regular user. MARGIN rejects few of the regular users' requests (by issuing puzzle or dropping them) at the beginning but after a while learns to successfully handle those requests by issuing *No Defense*



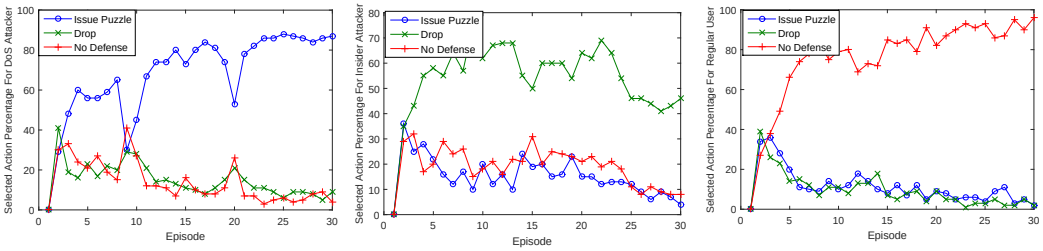
(i) DoS Attacker (ii) Regular User

(a) **Countermeasure Learning in Scenario 1**



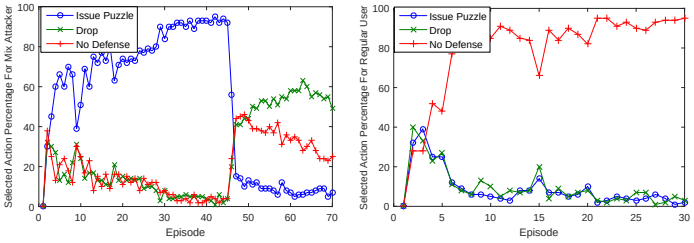
(i) Insider Attacker (ii) Regular User

(b) **Countermeasure Learning in Scenario 2**



(i) DoS Attacker (ii) Insider Attacker (iii) Regular User

(c) **Countermeasure Learning in Scenario 3**

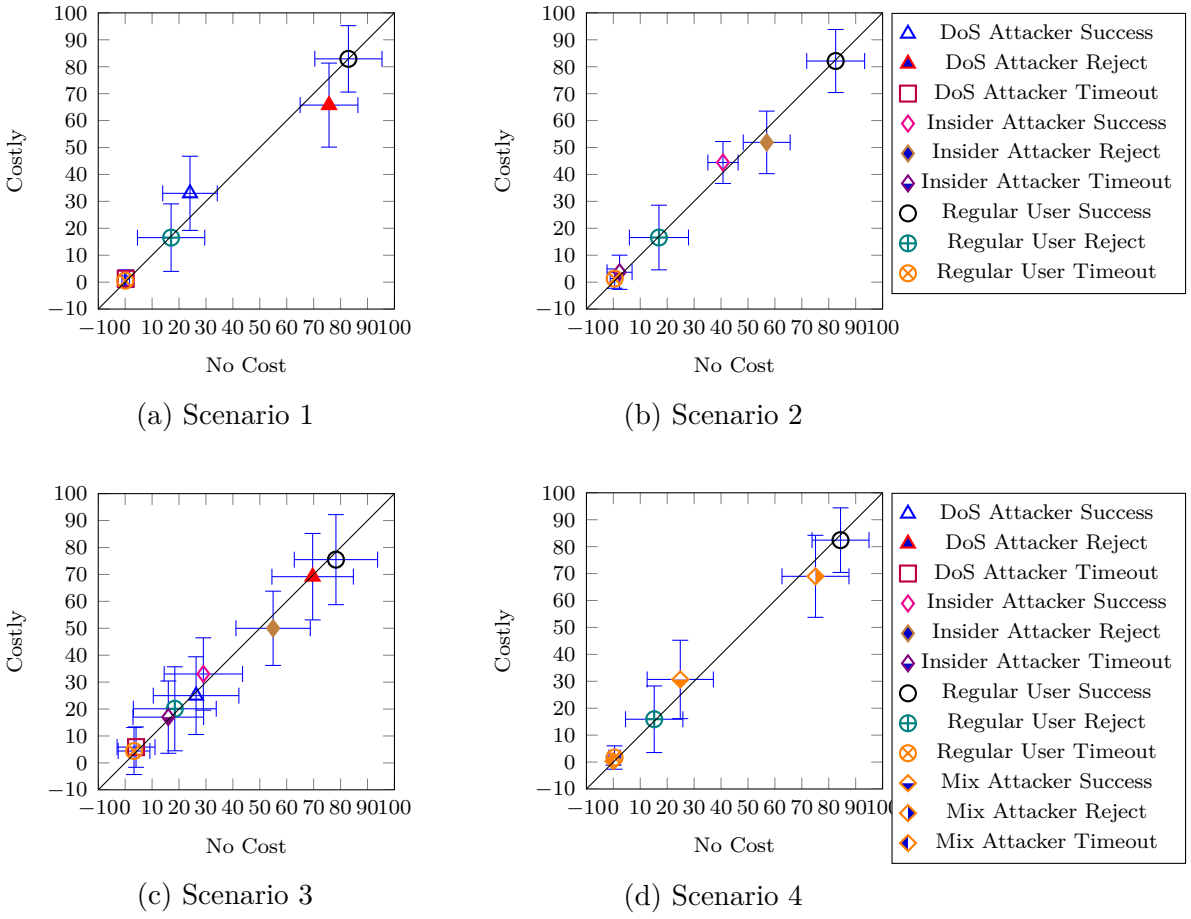


(i) Mix Attacker (ii) Regular User

(d) **Countermeasure Learning in Scenario 4**

Figure 5.3: RQ2.1 Scenarios

response action. Fig. 5.3c depicts the results when both attacks happen simultaneously (scenario 3). It shows that MARGIN learns (as desired) to trigger *Issue Puzzle* in case of *DoS* attack and trigger *Drop* in case of *Insider* attack. Fig. 5.3d shows that when both attacks happen sequentially (scenario 4), MARGIN is able to distinguish the change of the type of attack issued by the same attacker that is called *mix attacker* (case (i)) while the requests from the regular user are successfully handled (case(ii)) in more than 80% of times in average.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 5.4: RQ2.2 – No Cost Countermeasures vs. Costly Countermeasures

5.6.2 RQ2.2: What is the effect of the cost of a countermeasure?

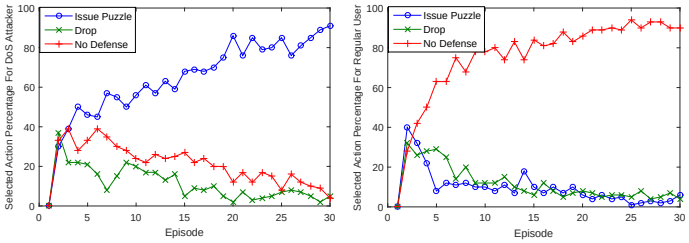
In the previous set of experiments, we assumed that the cost of countermeasures is not significant. Here, we kept the same parameters as in RQ2.1, except that we introduce a positive cost for the countermeasures. The cost is employed in the reward function as shown in Equation 3. To understand the effect of a countermeasure's cost in the proposed decision model, we compare the results for costly countermeasures with the results of no cost countermeasures.

Fig. 5.6 reveals that the model which uses action cost rejects fewer requests as *Drop* and *Issue Puzzle* countermeasures introduce cost to the system. Hence, the number of successful requests for DoS attacker (Scenario 1) and Insider attacker (Scenario 2) is increased while the number of rejected requests for the attackers is decreased compare to the model without action cost. Similarly for scenario 3 and 4, the rejection of attackers' requests is higher than in the case of *No Cost*. As a result, we can argue that there is a trade off between satisfying security goals and cost of countermeasures (more secure system with higher operational cost or less secure with lower cost).

Fig. 5.5a shows the result of costly countermeasures in terms of the percentage of actions selected for scenario 1. Compare to the result of previous research question, in which the countermeasures had no cost (Fig. 5.3a), it is obvious that *No Defense* response action is selected more than *Drop* response action when the system is under DoS attack. Similar results are achieved for scenario 2, 3. Likewise, the results for scenario 4 show a comparable outcome. As shown in Fig. 5.5d, when the type of attack changes, MARGIN first attempts to select *No Defense* as a response to this change and subsequently it learns the proper countermeasure for the undergoing type of attack.

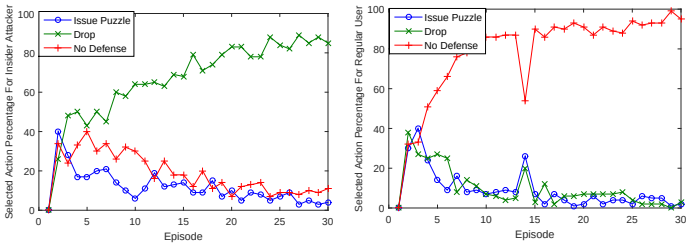
5.6.3 RQ2.3: What is the impact of the explore rate?

Here, we study the impact of the explore rate by comparing the result of two significantly different explore rates ($\epsilon = 0.1$ and $\epsilon = 0.9$) while the other parameters in MARGIN are kept the same ($\alpha = 0.5$ and $\gamma = 0.5$). Fig. 5.6 illustrates the results for these two values for the four attack scenarios. In all four scenarios, when $\epsilon = 0.1$, the outcome is more pleasant as the percentage of rejected requests for the attackers is higher while the percentage of successfully handled requests for the regular user is also higher.



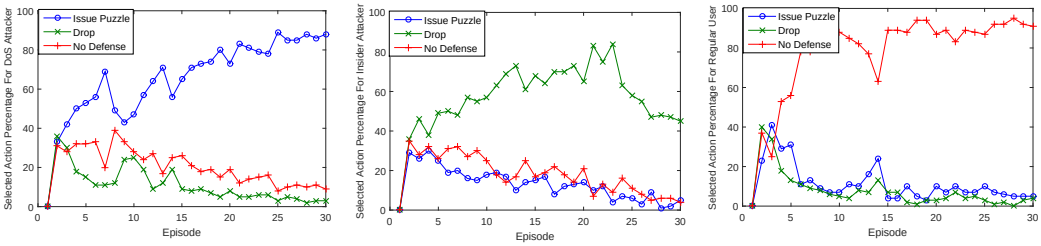
(i) DoS Attacker (ii) Regular User

(a) **Countermeasure Learning in Scenario 1**



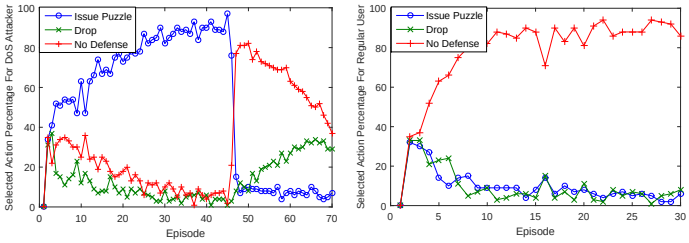
(i) Insider Attacker (ii) Regular User

(b) **Countermeasure Learning in Scenario 2**



(i) DoS Attacker (ii) Insider Attacker (iii) Regular User

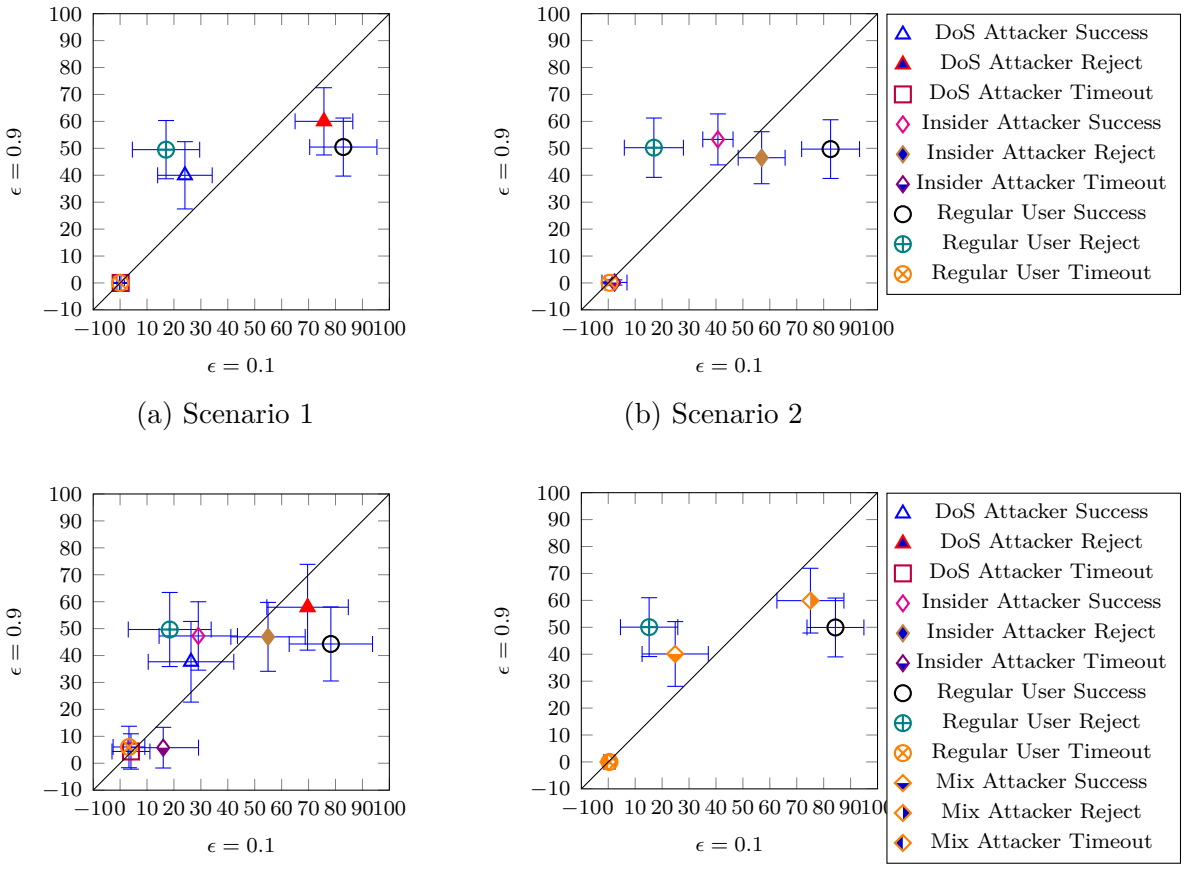
(c) **Countermeasure Learning in Scenario 3**



(i) Mix Attacker (ii) Regular User

(d) **Countermeasure Learning in Scenario 4**

Figure 5.5: RQ2.2 Scenarios



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

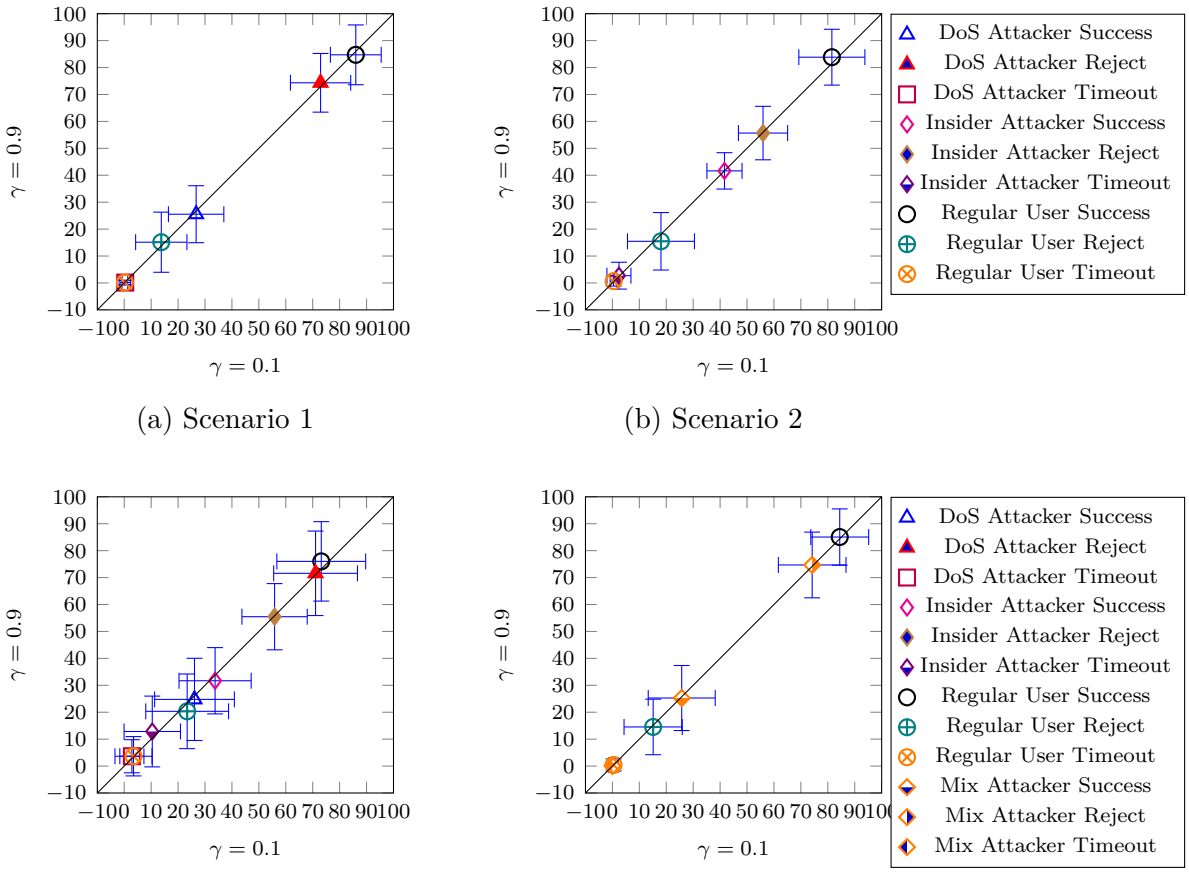
(d) Scenario 4

Figure 5.6: RQ2.3 – Comparing explore rates $\epsilon = 0.1$ vs. $\epsilon = 0.9$

These set of experiments generate some interesting results; when the ϵ value increases, the quality of self-protection declines as the success rate of attackers increases, and the success of regular users decreases as more regular users' requests and fewer attackers' requests are rejected. High ϵ values result in the system dynamically exploring different response actions and not triggering the proper countermeasure learned by the decision model.

5.6.4 RQ2.4: What is the effect of the discount factor?

In this set of experiments, we explored two significantly different values for the discount factor: $\gamma = 0.1$ and $\gamma = 0.9$. The rest of the parameters are the same



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

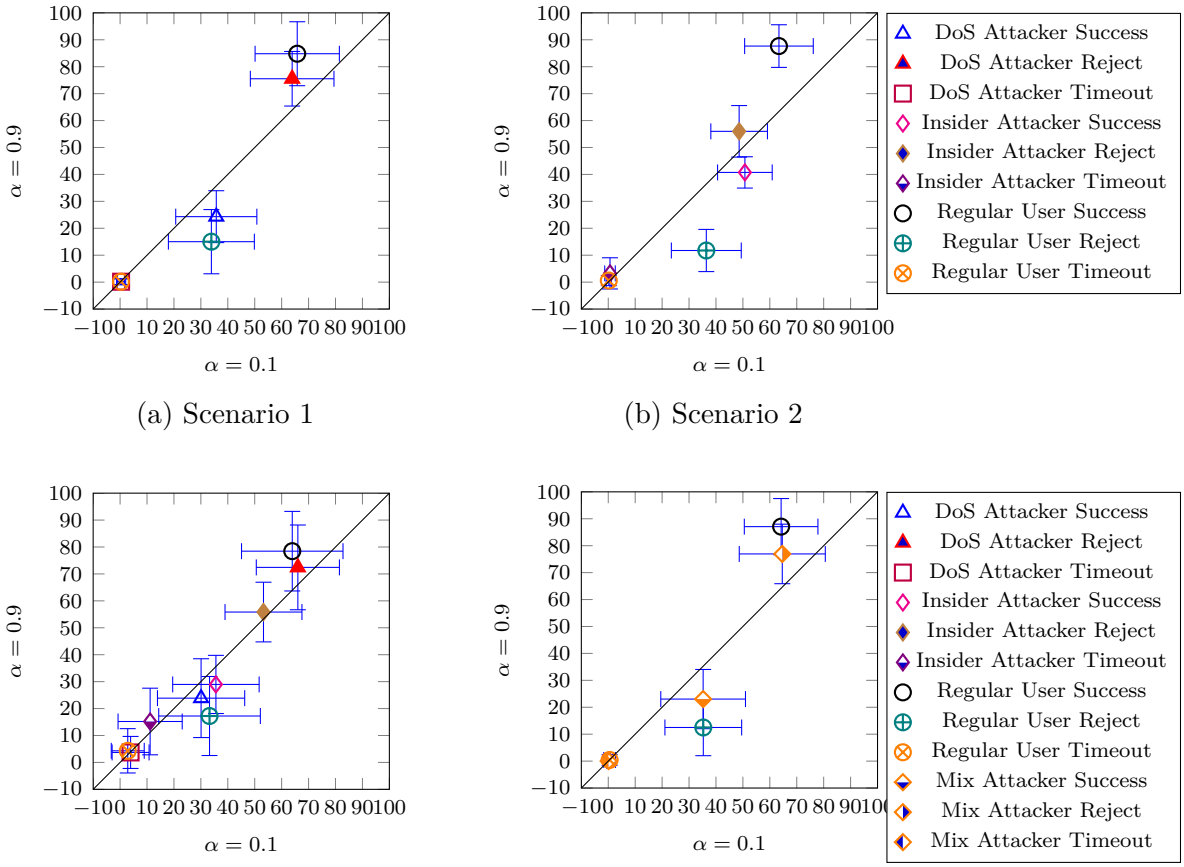
Figure 5.7: RQ2.4 – Comparing discount factors $\gamma = 0.1$ vs. $\gamma = 0.9$

as in the experiments in *RQ2.1*. Fig. 5.7 shows the results for these two values in the four attack scenarios. The results indicate no significant difference in the percentage values of success, reject, and timeout for attackers and regular users. This is consistent in all four studied scenarios.

The two studied values for the discount factor ($\gamma = 0.1$ and $\gamma = 0.9$) represent the near-term gain versus later rewards. Thus future rewards do not significantly affect decision making.

5.6.5 RQ2.5: What is the effect of the learning rate?

To evaluate the effect of the learning rate, we set its value to 0.1 and 0.9 and compared the outcome. The rest of the parameters are kept the same as in *RQ2.1*. Fig. 5.8 show the results for these two values in four studied attack scenarios.

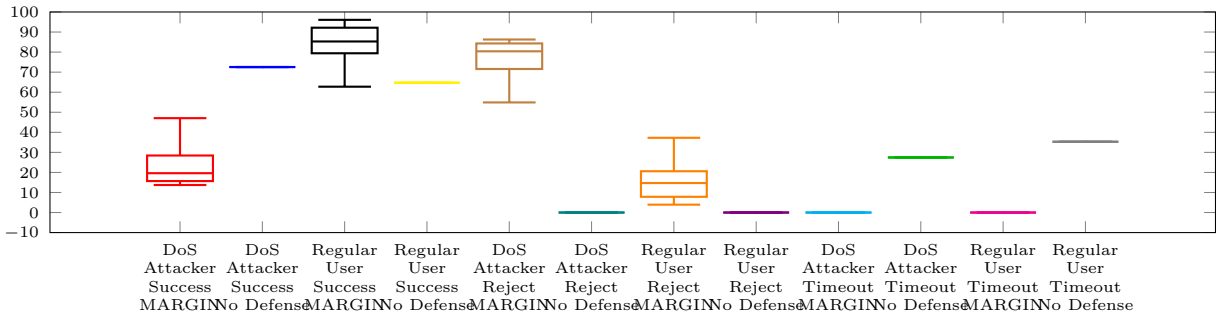


(a) Scenario 1 (b) Scenario 2

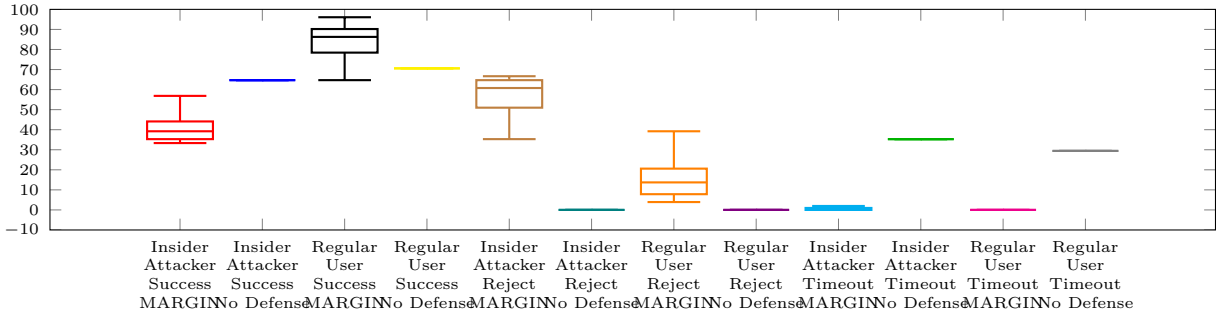
(c) Scenario 3 (d) Scenario 4

Figure 5.8: RQ2.5 – Comparing learning rates $\alpha = 0.1$ vs. $\alpha = 0.9$

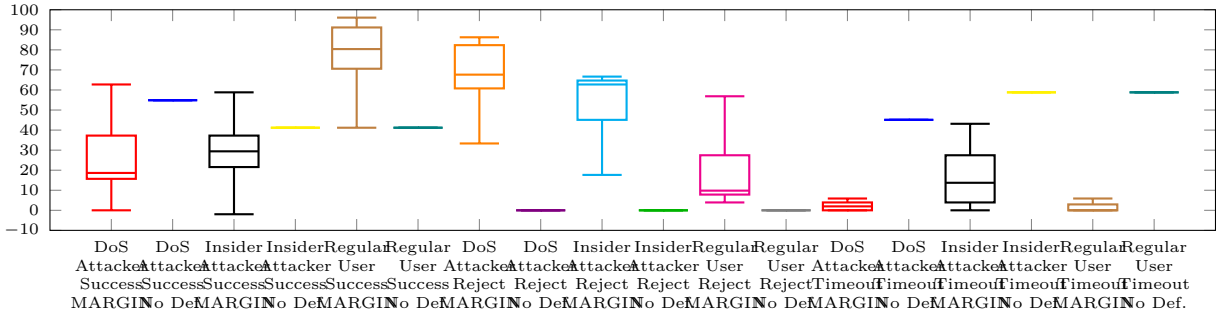
Comparison of the percentage values reveals that when the learning rate is set to a low value ($\alpha = 0.1$), attackers are more successful as the percentage of their rejected requests decreases while the percentage of regular users' successful requests also decreases. All the four attack scenarios consistently show that the most-desired result is achieved when the learning rate is set to a high value ($\alpha = 0.9$). Hence, as expected, low learning rate value results in failing to learn the proper countermeasure.



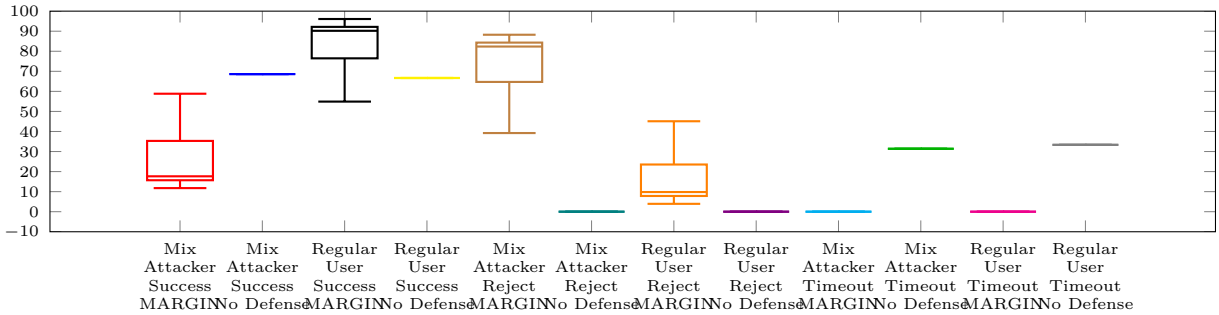
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



100

(d) Scenario 4

Figure 5.9: RQ2.6 – MARGIN vs. No Defense Box-plots

5.6.6 RQ2.6: Is learning a good idea? (How does the proposed technique perform comparing with other techniques?)

Here, we aim at evaluating whether the proposed MARGIN addresses attack-type uncertainty with reasonable performance compared to other approaches. Since the problem of attack-type uncertainty is not studied specifically in the SPS literature, we compare our approach to three other possible approaches, namely: (i) *No Defense*, (ii) *Random* action selection, and (iii) *Fixed* action selection. In the latter approach, a fixed countermeasure is applied to tackle attack-type uncertainty cases. The fixed countermeasure could be either *Issue Puzzle* or *Drop*.

In this set of experiments, we first compare the efficiency of MARGIN approach to that of not having a decision model in place (No Defense). Fig. 5.9 depicts box-plot diagrams of 100 executions for each technique. The percentage values of how the requests are handled in cases of *MARGIN* and *No Defense* are reported. On each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles. In the case of *No Defense*, the results for all 100 run are the same because the system always chooses the same response action – *No Defense* – and other response actions are not explored. As a result, no request is rejected despite its maliciousness.

For Scenario 1 (Fig. 5.9 (a)), MARGIN is capable of intelligently learning and applying the best countermeasure. Hence, high percentage of attack requests are rejected while high percentage of regular requests are successfully handled. Similar results can be noted for Scenario 2 in Fig. 5.9 (b). Fig. 5.9 (c) and (d) reveal that MARGIN can successfully handle multiple attacks, whether simultaneous or one after another. In all four scenarios, when there is no defense, some of the requests are timed out due to ongoing attack/attacks. The number of timed-out requests is decreased substantially using MARGIN as most attackers' requests are rejected by the SPS system. Therefore, the number of successful requests for DoS/insider attackers is much lower compared to the case of *No Defense*, while the number of successful requests for regular users is higher.

To validate the observations, these two techniques are compared using the Kruskal-Wallis one-way analysis of variance [72] . The choice of Kruskal-Wallis one-way analysis is because our datasets do not have equal variances and do not follow a normal distribution. Table 5.4 shows the results for the chi-square test and p-value. These findings indicate that in the significant level of 1% the MARGIN and No Defense are significantly different from each other. To scrutinize the results, it is essential to

look at the box-plots and pairwise comparison of the treatments.

Random action selection and MARGIN techniques are compared using the Kruskal-Wallis statistical test. Table 5.4 lists the results of the test for success, reject, and timeout requests for the attacker and regular user. In the significant level of 1%, these two techniques are different from each other. Comparing MARGIN technique with Random technique, the number of timeout requests in scenario 1 and scenario 4 for both attacker and regular users indicate less significant difference . However, the values for both of these two techniques are close to 0% and are not significant (as illustrated in their related box-plot diagrams in Fig. 5.10).

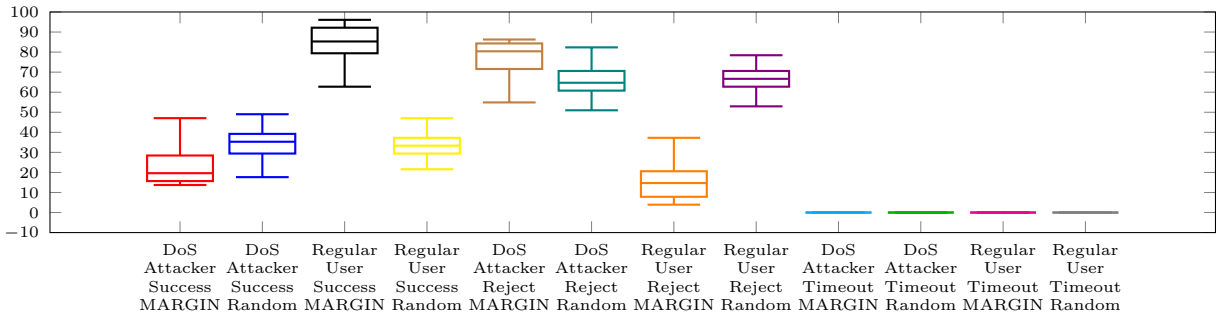
Moreover, the results for Random action selection (in Fig. 5.10) demonstrate that all users are treated similarly despite their maliciousness. Since the action selection is based on a random action out of a set of 3 actions, in average 66% of the requests are rejected (faced Issue Puzzle or Drop countermeasure) and 33% of the requests are successfully handled (No Defense countermeasure was applied). Box-plots in Fig. 5.10 as well as statistical comparisons in Table 5.4 indicate that applying MARGIN technique can significantly boost the efficiency of decision-making process compare to applying random action selection techniques.

In the case of Fixed action selection, both *Issue Puzzle* and *Drop* countermeasures reveal very similar results. Therefore, the results for one of them (*Issue Puzzle*) is presented here. Fixed action selection and MARGIN techniques are investigated using the Kruskal-Wallis statistical test in Table 5.4. The results indicate that these two techniques are statistically significant. The outcome of Fixed action selection (in Fig. 5.11) reveals that all requests are rejected in all scenarios despite their maliciousness. This can highly affect the legitimate users' experience and lowers the usability of the system. Accordingly, since at the time of decision making, the SPS system is not aware of the attack type, Fixed techniques are not considerate a practical choice.

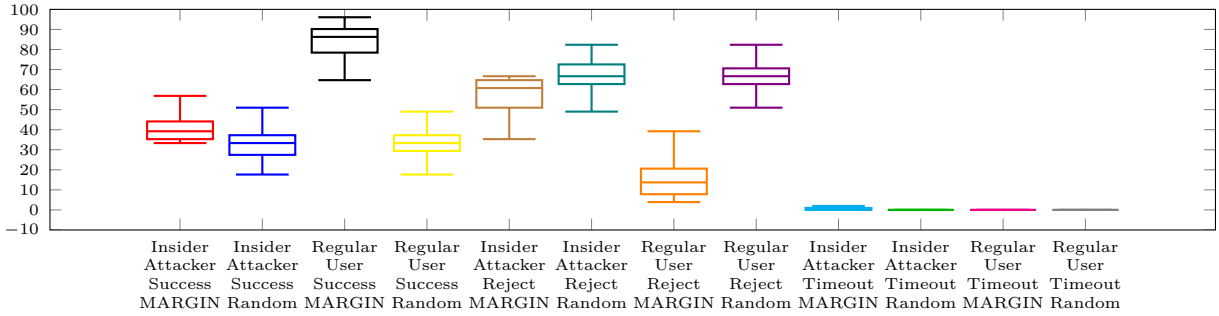
The experimental results show that MARGIN improved the SPS systems' ability to effectively handle attack-type uncertainty. It decreases service provision to malicious requests and increases handling legitimate requests during an ongoing attack. The response to *RQ2.6*, based on the obtained results, is that MARGIN improves system's self-protection at the time of attack-type uncertainty. MARGIN provides a principled approach to making rational decisions in the face of uncertainty about the type of attacks that target SPS systems.

Table 5.4: Kruskal-Waillis Test Results for RQ2.6

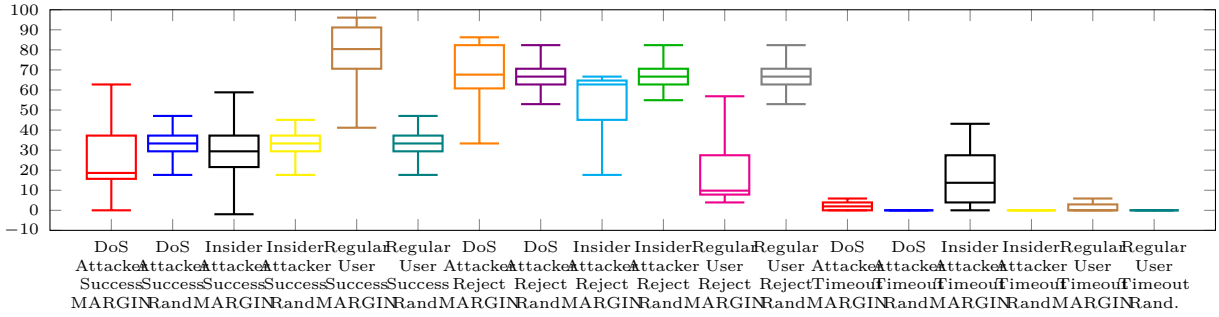
Scenario	**Comparison**	**No Defence**		**Random**		**Fix**	
		chi-square	p-value	chi-square	p-value	chi-square	p-value
Scenario 1	DoS Attacker Success	171.11	< 0.0001	52.88	< 0.0001	171.11	< 0.0001
	Regular User Success	112.59	< 0.0001	145.74	< 0.0001	170.86	< 0.0001
	DoS Attacker Reject	171.11	< 0.0001	58.7	< 0.0001	171.11	< 0.0001
	Regular User Reject	170.88	< 0.0001	145.75	< 0.0001	170.88	< 0.0001
	DoS Attacker Timeout	196.15	< 0.0001	3.03	0.08	3.03	0.08
	Regular User Timeout	197.07	< 0.0001	2.01	0.15	2.01	0.15
Scenario 2	Insider Attacker Success	171.18	< 0.0001	51.50	< 0.0001	171.18	< 0.0001
	Regular User Success	93.57	< 0.0001	148.66	< 0.0001	170.87	< 0.0001
	Insider Attacker Reject	171.09	< 0.0001	55.68	< 0.0001	171.09	< 0.0001
	Regular User Reject	170.96	< 0.0001	148.72	< 0.0001	170.96	< 0.0001
	Insider Attacker Timeout	181.52	< 0.0001	28.26	< 0.0001	28.26	< 0.0001
	Regular User Timeout	188.98	< 0.0001	12.68	< 0.0004	12.68	< 0.0004
Scenario 3	DoS Attacker Success	121.65	< 0.0001	18.66	< 0.0001	165.58	< 0.0001
	Insider Attacker Success	83.11	< 0.0001	6.95	< 0.009	161.26	< 0.0001
	Regular User Success	148.29	< 0.0001	137.20	< 0.0001	170.71	< 0.0001
	DoS Attacker Reject	170.93	< 0.0001	4.22	0.04	170.93	< 0.0001
	Insider Attacker Reject	171.22	< 0.0001	50.43	< 0.0001	171.22	< 0.0001
	Regular User Reject	171.05	< 0.0001	137.65	< 0.0001	171.05	< 0.0001
	DoS Attacker Timeout	173.72	< 0.0001	68.08	< 0.0001	68.08	< 0.0001
	Insider Attacker Timeout	170.81	< 0.0001	121.87	< 0.0001	121.87	< 0.0001
	Regular User Timeout	176.94	< 0.0001	44.57	< 0.0001	44.57	< 0.0001
Scenario 4	Mix Attacker Success	171.18	< 0.0001	34.28	< 0.0001	171.18	< 0.0001
	Regular User Success	126.62	< 0.0001	149.92	< 0.0001	171.20	< 0.0001
	Mix Attacker Reject	171.18	< 0.0001	34.28	< 0.0001	171.18	< 0.0001
	Regular User Reject	171.33	< 0.0001	150.03	< 0.0001	171.33	< 0.0001
	Mix Attacker Timeout	197.07	< 0.0001	2.01	< 0.15	2.01	0.15
	Regular User Timeout	193.54	< 0.0001	6.15	< 0.01	6.15	0.01



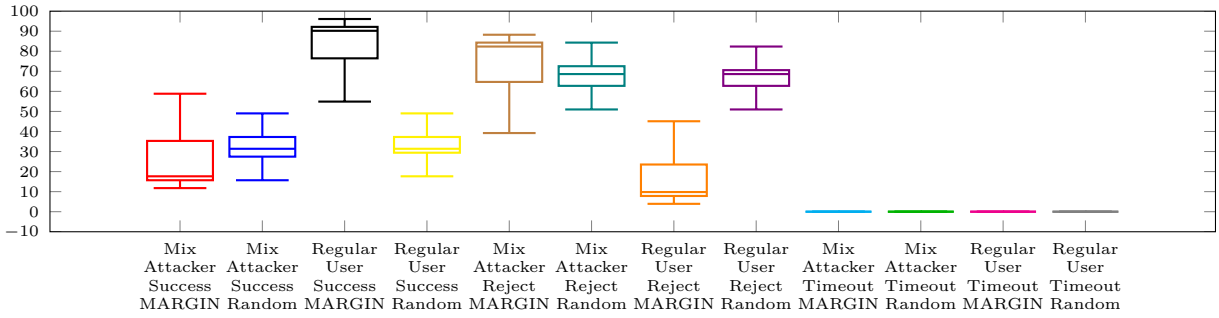
(a) Scenario 1



(b) Scenario 2

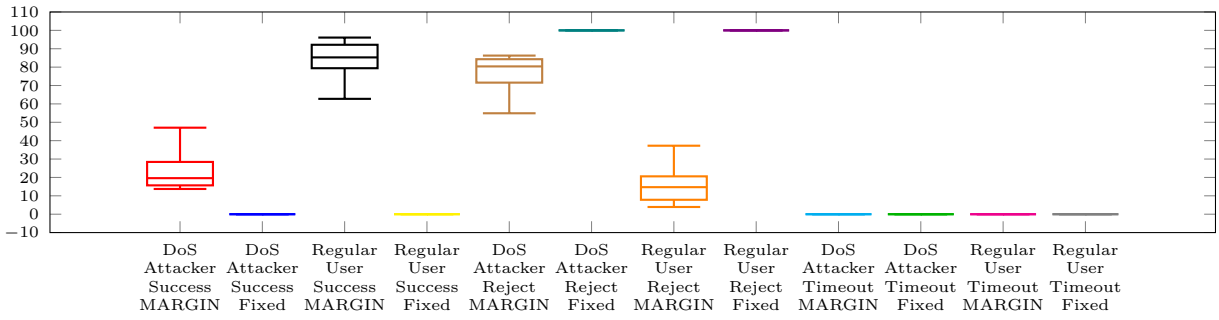


(c) Scenario 3

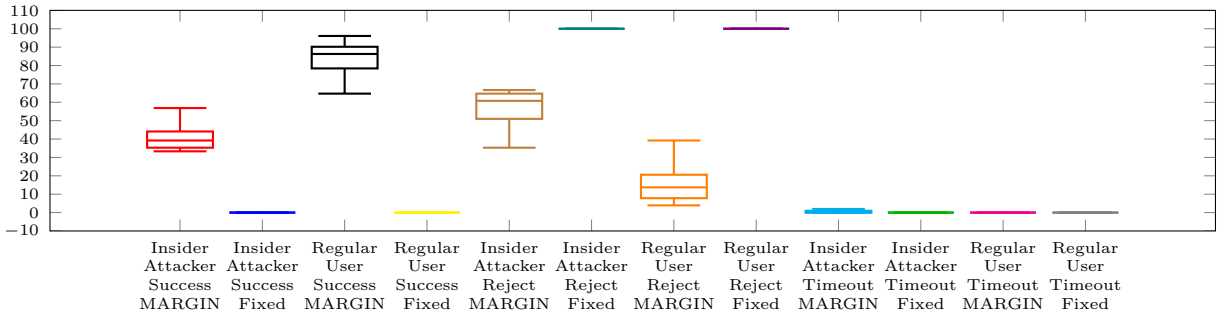


(d) Scenario 4

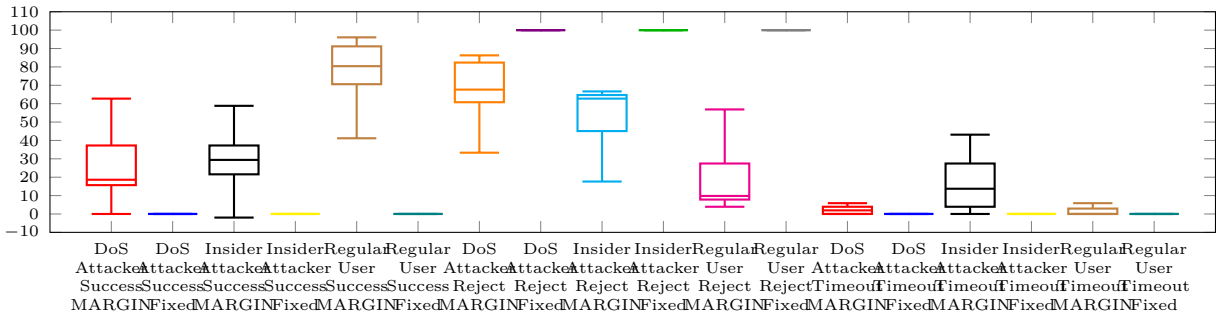
Figure 5.10: RQ2.6 – MARGIN vs. Random Box-plots



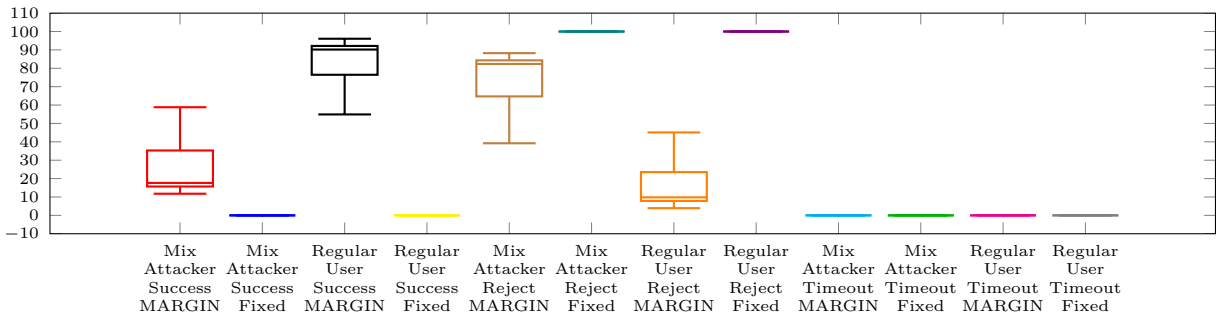
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



(d) Scenario 4

Figure 5.11: RQ2.6 – MARGIN vs. Fixed Box-plots

5.6.7 Threats to internal and external validity

The main threats to the validity of this research are as follows:

• **A Zero-sum Game:** The decision making assumes that the gain of one player is equal to the lost of the other player. Quantified harm caused by the attacker in many cases is not necessarily the same as the benefit gained by the attacker.

• **Rational Players:** An important underlying assumption of the proposed game-theoretic approach is the rationality of the decision-making engine and the attacker. However, they may not act rationally in reality, mainly because of limited observations and available information. In a two-player zero-sum game, if a defender chooses a Nash equilibrium strategy, the rationality of the attacker is not important. We should take into account that any deviation from the Nash equilibrium decreases the cost of the defender while increasing the the benefit of the attacker [4].

• **Trial-and-error Nature:** Learning-based algorithms have a trial-and-error nature. Thus at the early stages of each attack, the SPS system can make wrong decisions. The outcome of these decisions will guide the adaptation manager towards selecting the proper action in similar situations. Therefore, when the adaptation manager faces a new strategy, it requires learning time. This phenomenon may not be accepted in critical software systems that do not tolerate wrong decisions. The solution to this problem can be (i) considering a testing phase for the system as learning time, or (ii) initializing the payoff tables using expert knowledge.

Despite of aforementioned threats to validity, our game-theoretic analysis can provide a rational guideline for addressing attack-type uncertainty using a learning-based multi-agent approach (Markov game).

5.7 Summary

This chapter extends the proposed plug-and-play framework to employ Markov game technique aiming at addressing attack-type uncertainty in SPS systems. Uncertainty in this respect can be caused by (i) attacks that are well-planned such that they are indistinguishable from legitimate requests, or (ii) attacks that have the same scenario but target different quality goals of the SPS system. We presented a Markov game decision-making approach that learns about the type of attack in order to select the best countermeasure against it. For this purpose, this work focuses on learning the type of attack using the positive/negative impact of the applied countermeasure on

the quality goals. We utilize a learning technique, known as minimax-Q [82]. In order to aid in the gradual improvement of the SPS system's decision quality, we propose a novel and intuitive reward function that deals with the relations among quality goals, actions, and system attributes. A comprehensive set of experiments have been conducted in a simulation environment. Various possible scenarios are modeled in this environment and the results have demonstrated that the proposed MARGIN is capable of learning to select a proper countermeasure when facing attack-type uncertainty. The next chapter focuses on addressing attack-type uncertainty from a different perspective; Instead of a learning-based approach, a probabilistic-based approach is taken.

Chapter 6

Decision Making using Bayesian Games

We present a Bayesian game model that captures the uncertainty about an adversary's motivation for sending malicious requests. Our game-theoretic model formalizes possible intentions of adversaries along with the security preferences of the software system. In such a novel design, the equilibrium of the modeled game balances the gain from achieving security goals with the loss incurred by mitigating the attack. We provide an extensive analysis of the proposed game-theoretic model in the presence and absence of uncertainty about the adversary type. Moreover, we present a case study to show how such uncertainty can be addressed using the proposed technique in a real-world scenario.

Preserving the security goals of software systems when confronted with application-layer attacks requires not only detecting the attack, but also responding in a timely and appropriate manner. A manual intrusion response introduces a delay between notification and response, which could be exploited by the adversary to significantly increase damage to the system [137]. Protecting software systems against today's sophisticated attacks calls for intelligent decision-making techniques. However, the decision-making process needs to deal with various unknown features such as uncertainty associated with the satisficement of Non-Functional Requirements (NFRs) given a set of decisions [10] or uncertainty in alert notification [137].

In this chapter, we explore the incorporation of such uncertainty into the decision model of Self-Protecting Software (SPS) systems. We propose a Uncertainty-Based Self-Protection (UBSP) approach that can be easily employed in any SPS system according to its preferences in security goals and the cost of response actions while

fusing the cost and benefit of initiating attacks by various types of adversaries. The decision model is based on a Bayesian game model which acknowledges the interactions between the SPS system and the adversary. The decision model considers the uncertainty about the type of adversary by incorporating the probability for each type of adversary. We analyze the achievable Nash equilibrium for both pure and mix strategies for the SPS system and the rational adversary.

The rest of this chapter is organized as follows. Section 6.1 provides the notations used throughout this chapter. Section 6.2 elaborates on the detail of the proposed game that we investigate in this chapter. Section 6.3 presents the modeling technique in the extended framework. Section 6.4, and 6.5 defines two normal-form games with complete information about the type of adversary as part of the designing phase of the decision-making engine. Section 6.6 completes the designing phase of the game by describing a Bayesian game model to formulate the decision model with uncertainty about the type of adversary. Section 6.7 realizes the defined Bayesian game by a case study on a telephony system. Section 6.8 discusses the obtained results of the case study. Finally, Section 6.9 summarizes the chapter.

6.1 Notations

In our modeled games, two players choose strategies simultaneously under the assumption that both have common knowledge about the cost and benefits of the game. Table 6.1 summarizes the notations used in our UBSP game model: C_x denotes the cost of actions for the players (the adversary and the defense system), M_{ATj}^{Ti} represents the gain by the malicious user (adversary), and S_{Gi} denotes the gain by the SPS system. For both players, all possible strategies incur some cost, which can be interpreted as the operational cost to conduct the strategy. The benefits of goals G_1 and G_2 are represented by S_{G1} and S_{G2} and are defined by the stakeholders.

In the defined game model, the decision factors are abstract enough to adapt to the required application. For example, the values of S_{G1} and S_{G2} can be dependent on the type of application and the preferences over security goals in the system. In Table 6.1, it is reasonable to assume: (i) $S_{G1}, S_{G2} > C_{CM1}$, (ii) $S_{G1}, S_{G2} > C_{CM2}$, (iii) $M_{AT1}^{T1}, M_{AT2}^{T1} > C_{AT1}$, and (iv) $M_{AT1}^{T1}, M_{AT2}^{T1} > C_{AT2}$, since otherwise the defense system does not have the incentive to defend the system and the adversary does not have the incentive to attack. The cost of response actions (C_{CM1} and C_{CM2}) can be defined as a function of resource consumption with respect to activities to enable and process the response action. The cost of AT1 (C_{AT1}) and AT2 (C_{AT2}) can be

Table 6.1: Summary of Notations Used

Notation	Definition
a	Adversary (Player)
d	Defense System (Player)
ATi	Attack i (Action)
CMi	Countermeasure i (Action)
C_x	Cost of applying action x
Ti	Adversary type i
M_{ATj}^{Ti}	Measure of benefit to the Ti type adversary to achieve its malicious goal via ATj
Gi	Security goal i
S_{Gi}	Measure of benefit to the defense system to satisfy security goal Gi
p	Probability with which the defense system plays CM1
$1-p$	Probability with which the defense system plays CM2
q_i	Probability with which the type i adversary plays AT1
$1-q_i$	Probability with which the type i adversary plays AT2
μ	Probability of an adversary of type 1
$1-\mu$	Probability of an adversary of type 2
$E_{Player}(Action)$	Expected payoff of a $Player$ to play an $Action$

defined as a function of resource consumption to initiate the attack.

6.2 A Bayesian Game Approach

We address the uncertainty about the type of adversary by incorporating such information, along with the benefits and costs of attacks and countermeasures, into the

decision model. We consider a two-player normal-form game in which, one player is the adversary, denoted by "a". The other player is the defense system in the SPS, denoted by "d". In this game, the adversary aims at targeting one of the two security goals of the system: G_1, and G_2. For instance, the aim of the adversary could be breaking down the server in order to discount the *availability* goal (G_1) of the SPS system or to access sensitive information and break the *confidentiality* goal (G_2). Each targeted security goal calls for a different type of adversary. For example, the availability goal can be targeted by Denial of Service (DoS) attackers whereas the confidentiality goal is targeted by insider attackers.

Today's sophisticated and well-planned attack scenarios have motivated us to consider attack scenarios in which the adversary adapts more than one pure strategy to mislead the defense system. In this case, a traditional defense system may select a countermeasure that works to the benefit of the adversary. The benefits of game theory in providing a holistic decision making in adaptive security is discussed in [43]. In our UBSP game specification, the SPS system protects two security goals: G_1 and G_2. Accordingly, two pure strategies for the adversary are considered: *Attack 1* (AT1) and *Attack 2* (AT2), which target G_1 and G_2, respectively. However, an intelligent adversary may mix these two strategies, aiming at confusing the SPS system. For example, a DoS attacker can mix two pure strategies: (i) *Heavy Load*: sending malicious requests that introduce high workload to the system, and (ii) *High Sensitive*: sending malicious requests that target files with low workload, yet sensitive data. The former strategy is prone to fast detection by the IDS, whereas the latter strategy results in misdiagnosing the attack and applying a countermeasure that is effective for an insider attacker instead of a DoS attack. Therefore, the defense system will fail to treat the adversary as a DoS attacker.

In our UBSP game specification, the defense system has two pure strategies to protect the software system: *Countermeasure 1* (CM1) and *Countermeasure 2* (CM2), which protect G_1 and G_2, respectively. For example, two possible countermeasures are (i) *Issue Puzzle*: issuing a puzzle such as CAPTCHA [126] to determine whether or not the original of the request is from a real user or a botnet [69], and (ii) *Drop Request*: providing no respond to the incoming request. These two countermeasures protect the availability and confidentiality goals, respectively. In many situations, the defense system can only select and apply one of the available countermeasures to protect the software system due to the fact that: (i) most of the countermeasures are mutually exclusive, (ii) the combined cost of applying countermeasures exceeds the value of the software protected, or (iii) the combination of countermeasures degrades the level of service to an unacceptable level.

6.3 Modeling: Cost and Benefits of Strategies

In the modeling phase, the objective is to facilitate defining payoffs of the players. Accordingly the satisfaction of the defense system's (or the adversary's) quality (or malicious) goal is considered along with the cost of applying each particular strategy (a countermeasure or an attack). In other words, the payoffs of the players for each strategy is defined by measuring the benefit of the strategy for the player (the defense system or the adversary) minus the cost of applying that strategy.

The benefit of a strategy is quantified by the impact of that strategy on the quality/malicious goal of the defense system/the adversary. Incorporating such information into the modeling the interdependencies of strategies (between the SPS system and the adversary) is proposed in [44]. Here, we simply formulate this information into the payoff values. The following values are incorporated: (i) The level of satisfaction of a security goal for the defense system (S_{Gi}), (ii) The level of achievement of a malicious goal for the adversary (M_{ATj}^{Ti}), (iii) The cost of triggering a strategy for a player (C_x). Based on these information, the following sections describe two normal-form games [96] with complete information about the type of adversary. In each game, we model the payoffs of players and analyze the Nash equilibrium solution of the game along with a numerical example.

6.4 Designing: Type 1 Adversary

In this complete information game, we consider the scenario in which the intention of the adversary is to target the G_1 security goal (a type 1 adversary). Hence, the adversary initiates AT1 along with AT2 to mislead the other player (the defense system). Table 6.2 lists the payoffs. Each cell in Table 6.2 has two payoffs that correspond with player d and player a accordingly. The notations that represent the gain and cost of players are defined in Table 6.1.

For the SPS system, satisfaction of the two security goals G_1 and G_2 are denoted by S_{G1} and S_{G2}. The cost of applying CM1 is denoted by C_{CM1}. In the case of pure strategy AT1, applying CM1 can successfully address the attack and G_1 security goal is protected with this strategy. Therefore, the payoff of the SPS system is $S_{G1} - C_{CM1}$. Applying CM2 costs C_{CM2} and CM2 does not properly satisfy the security goal of G_1. For example, dropping the request will not stop the adversary from sending malicious requests. The continuation of such requests eventually results

Table 6.2: Strategic Form of Type 1 Adversary vs. SPS

d \ a	**AT1**	**AT2**
CM1	$S_{G1} - C_{CM1}$, $-M^{T1}_{AT1} - C_{AT1}$	$-S_{G2} - C_{CM1}$, $-M^{T1}_{AT2} - C_{AT2}$
CM2	$-S_{G1} - C_{CM2}$, $M^{T1}_{AT1} - C_{AT1}$	$S_{G2} - C_{CM2}$, $M^{T1}_{AT2} - C_{AT2}$

in threatening G_1 security goal (such as the availability goal). Hence, the payoff for CM2 in case of AT1 is $-S_{G1} - C_{CM2}$.

When an adversary wanting to target G_1 security goal issues AT2 in order to stay hidden from the IDS, the strategy CM1 puts security goal G_2 at risk, as represented by $-S_{G2}$. In this case, the response action of selecting CM1 results in a payoff of $-S_{G2} - C_{CM1}$, whereas selection of CM2 results in protection of security goal G_2 and the payoff of $S_{G2} - C_{CM2}$.

Now we turn to the adversary payoffs. The adversary incurs the cost of C_{AT1} by issuing AT1 and gains M^{T1}_{AT1} if the malicious intention to threat the G_1 security goal is satisfied. If the adversary fails, the gain of the adversary is $-M^{T1}_{AT1}$. Here, it can be seen that the success of an attack depends on the strategy selected by the SPS system. Only when SPS chooses CM2 will the adversary succeed ($M^{T1}_{AT1} - C_{AT1}$).

The adversary could also choose AT2 to misguide the SPS system and encourage it to change its strategy to CM2. The gain in this case is M^{T1}_{AT2}, which is less than M^{T1}_{AT1} because the adversary reaches its malicious goal quicker by issuing AT1 instead of AT2. Hence, we have: $M^{T1}_{AT1} > M^{T1}_{AT2}$. AT2 also introduces extra effort for the adversary that is intended to target G_1 security goal. This effort is denoted by C_{AT2}.

6.4.1 Nash Equilibrium Analysis

The well-known concept of Nash equilibrium [96] defines a set of actions for players such that none have any incentive to deviate from their chosen action. Assuming that the defense system always takes the pure strategy CM1, then the adversary's best response is to select AT2 when $-M^{T1}_{AT1} - C_{AT1} \leq -M^{T1}_{AT2} - C_{AT2}$. However, this is not an equilibrium, as the pure strategy of AT2 by the adversary motivates the rational defense system to change its strategy to CM2. By switching to CM2, the adversary is inclined to change its strategy to AT1 when $M^{T1}_{AT1} - C_{AT1} \geq M^{T1}_{AT2} - C_{AT2}$. Since

we have $M_{AT1}^{T1} > M_{AT2}^{T1}$, this can be interpreted thus: if the defense system applies CM2 then the adversary issues AT1 when $C_{AT1} \leq M_{AT1}^{T1} - M_{AT2}^{T1} + C_{AT2}$.

The above finding is interesting in the decision-making of SPS systems. It suggests that in selecting a response action, the decision-making engine needs to incorporate the cost of *initiating an attack* for adversaries rather than considering the cost of *applying a countermeasure* for the defense system. This is contrary to most approaches for SPS systems, where the decision-making engine considers only the operational cost of providing a response action. The application of such a decision-making engine can introduce a perspective system (e.g., by focusing on increasing the cost of mounting attacks for adversaries) rather than simply deploying a response action.

6.4.2 Mixed Strategy Equilibrium Analysis

In the previous subsection, we found that there are two pure strategies: (i) (CM1, AT1) when $-M_{AT1}^{T1} - C_{AT1} > -M_{AT2}^{T1} - C_{AT2}$, and (ii) (CM2, AT2) when $M_{AT1}^{T1} - C_{AT1} < M_{AT2}^{T1} - C_{AT2}$. Hence, there is no pure strategy Nash equilibrium when $-M_{AT1}^{T1} - C_{AT1} \leq -M_{AT2}^{T1} - C_{AT2}$ and $M_{AT1}^{T1} - C_{AT1} \geq M_{AT2}^{T1} - C_{AT2}$. Now, we check for Nash equilibrium when the adversary plays a mixed strategy. A *mix strategy* is randomizing over the set of available actions according to some probability distribution [115]. The expected payoffs of the defense system d are as follows (The notations that represent the probabilities (q_1, $1 - q_1$, p, and $1 - p$) are defined in Table 6.1)).

$$E_d(CM1) = q_1(S_{G1} - C_{CM1}) + (1 - q_1)(-S_{G2} - C_{CM1}), \tag{6.1}$$

$$E_d(CM2) = q_1(-S_{G1} - C_{CM2}) + (1 - q_1)(S_{G2} - C_{CM2}). \tag{6.2}$$

To make CM1 and CM2 indifferent to the defense system, i.e., $E_d(CM1) = E_d(CM2)$, the adversary's equilibrium strategy is to play AT1 with $q_1 = \frac{S_{G2} - C_{CM2} + S_{G2} + C_{CM1}}{2S_{G1} + 2S_{G2}}$. Similarly, the expected payoffs of the adversary are

$$E_a(AT1) = p(-M_{AT1}^{T1} - C_{AT1}) + (1 - p)(M_{AT1}^{T1} - C_{AT1}), \tag{6.3}$$

$$E_a(AT2) = p(-M_{AT2}^{T1} - C_{AT2}) + (1 - p)(M_{AT2}^{T1} - C_{AT2}). \tag{6.4}$$

By a similar calculation it can be shown that to make AT1 and AT2 indifferent to the adversary, i.e., $E_a(AT1) = E_a(AT2)$, the defense system's equilibrium strategy is to play CM1 with probability $p = \frac{M^{T1}_{AT2} - C_{AT2} - M^{T1}_{AT1} + C_{AT1}}{2M^{T1}_{AT2} - 2M^{T1}_{AT1}}$. It is noteworthy that deploying a countermeasure according to probability p is based on the given circumstances and has a risk of applying the wrong countermeasure.

Table 6.3: Payoffs and Numerical Examples of Type 1 Adversary

Table 6.3.a: Payoffs of Type 1 Adversary in G_1 Preferred SPS

d \ a	**AT1**	**AT2**
CM1	U^{++}, U^{--}	U^{-}, U^{-}
CM2	U^{--}, U^{++}	U^{+}, U^{+}

Table 6.3.c: Payoffs of Type 1 Adversary in G_2 Preferred SPS

d \ a	**AT1**	**AT2**
CM1	U^{+}, U^{--}	U^{--}, U^{-}
CM2	U^{-}, U^{++}	U^{++}, U^{+}

Table 6.3.b: Numerical Example of Type 1 Adversary in G_1 Preferred SPS

d \ a	**Heavy Load**	**High Sensitivity**
Issue Puzzle	$85, -100$	$-15, -90$
Drop Request	$-95, 85$	$5, 50$

Table 6.3.d: Numerical Example of Type 1 Adversary in G_2 Preferred SPS

d \ a	**Heavy Load**	**High Sensitivity**
Issue Puzzle	$5, -100$	$-95, -90$
Drop Request	$-15, 85$	$85, 50$

6.4.3 Case Based Analysis

Using the defined game model, in the following, we consider two possible cases: (i) an SPS system with a much stronger preference to protect G_1 security goal than G_2 security goal, and (ii) an SPS system with a much stronger preference to protect G_2 security goal than G_1 security goal. We provide a numerical example and analyze potential responses for each case.

- **Case 1: Type 1 Adversary vs. SPS System with G_1 Preference**

  Assuming the benefit to protect G_1 is much higher than that for G_2, we have: $S_{G1} >> S_{G2}$. The payoffs for the defense system d and the adversary a are as shown in Table 6.3.a. Here, U represents the outcome related to Table 6.2 while shows the the magnitude of the gain/lost in the payoff value.

Table 6.3.b exemplifies a type 1 adversary versus an SPS system with a high preference for G_1 security goal. The calculated mix strategy Nash equilibrium is: $p = 0.78$, $1 - p = 0.22$, $q_1 = 0.10$, and $1 - q_1 = 0.90$.

The resulting Nash equilibrium illustrates that in the mix strategy the response action CM1 has a much higher probability to be selected than the response action CM2. The rational adversary is motivated to attack with the probability of 0.10, due to the observation of mix strategy by the defense system d, which selects CM1 with the probability of 0.90. Hence, the rational adversary is discouraged from attacking the SPS system.

- **Case 2: Type 1 Adversary vs. SPS System with G_2 Preference**

  The second case considers a low preference of G_1 security goal and high preference of G_2 security goal ($S_{G1} << S_{G2}$). Payoffs for both players are shown in Table 6.3.c. Table 6.3.d provides a numerical example of such a scenario. We get a mix strategy Nash equilibrium for the game with $p = 0.78$, $1-p = 0.22$, $q_1 = 0.90$, and $1-q_1 = 0.10$. Accordingly, the rational response action for the defense system against AT1 is to choose CM1 with a much higher probability than CM2.

Note that the mix strategy response action selections (p & $1 - p$) in both cases are the same, because the defense system selects a response action by considering the goal of the adversary and the cost of the attack. In both cases, the intention behind the attacks and their cost are the same. Hence, the defense system takes the same strategy in protecting security goals. Comparing the two cases reveals that in case 1 the rational adversary chooses AT1 with less probability. This is motivated by the possibility of the defense system benefiting greatly id CM1 is chosen.

6.5 Designing: Type 2 Adversary

In the case of the adversary targeting the G_2 security goal, we model a normal form game in Table 6.4. The notations that represent the gain and cost of players are defined in Table 6.1. The payoffs for the defense system are based on the gain/lost of G_1 and G_2 security goals, and the cost of response actions accordingly. The payoffs for the player d in Table 6.4 are the same as in Table 6.2. However, in this scenario, the payoffs for the adversary differ from those in the game modeled in the previous section, due to the changed intention of the adversary.

In this game, the adversary gains/loses a value of M^{T2}_{AT2} if AT2 is successful/unsuccessful. Meanwhile the adversary may choose to issue AT1 in order to misguide

Table 6.4: Strategic Form of Type 2 Adversary vs. SPS

d \ a	**AT1**	**AT2**
CM1	$S_{G1} - C_{CM1}$, $M^{T2}_{AT1} - C_{AT1}$	$-S_{G2} - C_{CM1}$, $M^{T2}_{AT2} - C_{AT2}$
CM2	$-S_{G1} - C_{CM2}$, $-M^{T2}_{AT1} - C_{AT1}$	$S_{G2} - C_{CM2}$, $-M^{T2}_{AT2} - C_{AT2}$

the defense system. In this case, the gain/loss of applying this action is represented by M^{T2}_{AT1}. In the modeled game, the gain/loss of AT2 is higher than in the AT1 attack since the intention of the adversary is to break the G_2 security goal. Hence, we have $M^{T2}_{AT1} < M^{T2}_{AT2}$. As in the previous scenario, to motivate the rational adversary, we assume that $M^{T2}_{AT1}, M^{T2}_{AT2} > C_{AT1}, C_{AT2}$.

Table 6.5: Payoffs and Numerical Examples of Type 2 Adversary

Table 6.5.a: Payoffs of Type 2 Adversary in G_1 Preferred SPS

d \ a	**AT1**	**AT2**
CM1	U^{++}, U^{+}	U^{-}, U^{++}
CM2	U^{--}, U^{-}	U^{+}, U^{--}

Table 6.5.b: Numerical Example of Type 2 Adversary in G_1 Preferred SPS

d \ a	**Heavy Load**	**High Sensitivity**
Issue Puzzle	$85, 50$	$-15, 85$
Drop Request	$-95, -90$	$5, -100$

Table 6.5.c: Payoffs of Type 2 Adversary in G_2 Preferred SPS

d \ a	**AT1**	**AT2**
CM1	U^{+}, U^{+}	U^{--}, U^{++}
CM2	U^{-}, U^{-}	U^{++}, U^{--}

Table 6.5.d: Numerical Example of Type 2 Adversary in G_2 Preferred SPS

d \ a	**Heavy Load**	**High Sensitivity**
Issue Puzzle	$5, 50$	$-95, 85$
Drop Request	$-15, -90$	$85, -100$

6.5.1 Nash Equilibrium Analysis

Assuming that the defense system takes the pure strategy CM1, then the best response for the adversary is AT2 when $M^{T2}_{AT1} - C_{AT1} \leq M^{T2}_{AT2} - C_{AT2}$. However, if the

adversary plays AT2, then CM1 will not be the best response for the defense system, which will play CM2 instead. Hence, the adversary is motivated to trigger AT1 when $-M_{AT1}^{T2} - C_{AT1} \geq -M_{AT2}^{T2} - C_{AT2}$. Accordingly, pure strategy Nash equilibrium exists when: (i) $M_{AT1}^{T2} - C_{AT1} > M_{AT2}^{T2} - C_{AT2}$, or (ii) $-M_{AT1}^{T2} - C_{AT1} < -M_{AT2}^{T2} - C_{AT2}$.

6.5.2 Mixed Strategy Equilibrium Analysis

We check for Nash equilibrium when the adversary plays mix strategy AT1 with probability q_2 and AT2 with probability $1 - q_2$. The defense system's expected payoffs ($E_d(CM1)$ and $E_d(CM2)$) are the same as the expected payoffs defined in Equations (1) and (2). The adversary's equilibrium strategy is to play AT1 with $q_2 = \frac{S_{G2} - C_{CM2} + S_{G2} + C_{CM1}}{2S_{G1} + 2S_{G2}}$, similar to the case of the type 1 adversary in the previous section. This similarity suggests that despite the type of adversary, the probability of an attack is highly dependent on the security preference of the targeted system as well as the cost of applying a countermeasure. The expected payoffs of the adversary are

$$E_a(AT1) = p(M_{AT1}^{T2} - C_{AT1}) + (1 - p)(-M_{AT1}^{T2} - C_{AT1}), \tag{6.5}$$

$$E_a(AT2) = p(M_{AT2}^{T2} - C_{AT2}) + (1 - p)(-M_{AT2}^{T2} - C_{AT2}). \tag{6.6}$$

The defense system's equilibrium strategy is to make AT1 and AT2 indifferent to the adversary ($E_a(AT1) = E_a(AT2)$). Therefore, the defense system must choose CM1 with probability $p = \frac{M_{AT1}^{T2} + C_{AT1} - M_{AT2}^{T2} - C_{AT2}}{2M_{AT1}^{T2} - 2M_{AT2}^{T2}}$. As we have pointed out in Section 6.4.2, applying a countermeasure using probability, has the risk of applying the wrong countermeasure. However, considering the costs and benefits of actions in our analysis aims at decreasing such risk.

6.5.3 Case Based Analysis

In this section, we introduce two cases for the proposed game theoretic model and analyze the potential responses of the two players.

- **Case 1: Type 2 Adversary vs. SPS System with G_1 Preference**

Assuming that the defender benefits much more by keeping the G_1 security goal than the G_2 security goal, we have: $S_{G1} >> S_{G2}$. Given this assumption, the payoffs for both the defense system and the adversary are as shown in Table 6.5.a. Similar to Section 6.4.3, U represents the outcome related to Table 6.4 while shows the the magnitude of the gain/lost in the payoff value.

Table 6.5.b exemplifies an insider attack on an SPS system with a high preference for G_1 security goal. The equilibrium solution of the game is $p = 0.22$, $1-p = 0.78$, $q_2 = 0.10$, and $1 - q_2 = 0.90$. The adversary will rationally choose to initiate AT2 with the probability of 0.10. Therefore, the defense system's best response to the adversary's mix strategy is to select CM2 with the probability of 0.78.

- **Case 2: Type 2 Adversary vs. SPS System with G_2 Preference**

  Assume that the defender benefits much more from protecting security goal G_2 than from protecting security goal G_1. Hence, we have: $S_{G2} >> S_{G1}$. The payoffs are shown in Table 6.5.c. An example of payoffs for both players are shown in Table 6.5.d. In this case, we get a mix strategy equilibrium for the game when $p = 0.22$, $1-p = 0.78$, $q_2 = 0.90$, and $1-q_2 = 0.10$. Therefore, the rational action for the defense system is to choose CM1 only if it believes $q_2 > 0.90$. Otherwise, CM2 is the best response.

In the above two cases with different goal preferences, the probability of response action CM2 is identical in both cases, because the intention of the adversary in both cases is the same. In case 2, the probability of AT2 is lower than in case 1 due to its higher risk of benefiting the defense system.

In the two game models discussed in Sections 6.4 and 6.5, the defense system is able to make an intelligent decision that considers both (i) system security goals, and (ii) adversary intentions in targeting the SPS system. The type of adversary (its costs and benefits) is considered to be known when deciding on the response action. However, this is not always the case. Next, we discuss a game theoretic model that incorporates such uncertainty into the response action selection.

6.6 Designing: Adversary-Type Uncertainty

In this section, we consider scenarios in which the defender is uncertain about the intention of the adversary in targeting the SPS system. This situation could be interpreted as that of responding to two types of adversaries. Bayesian game technique

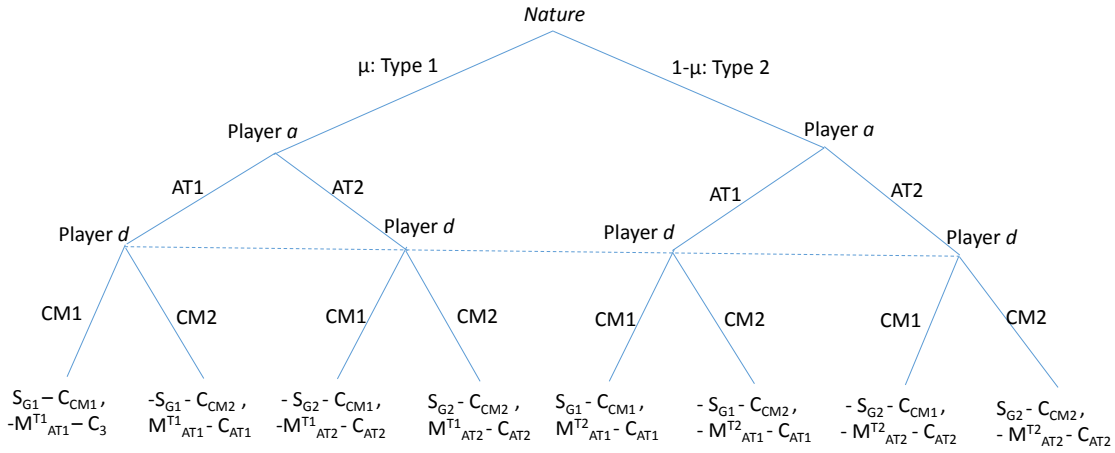


Figure 6.1: Extensive Form of the Modeled Bayesian Game

models the similar situation in which the player is uncertain about the type(s) of other player(s). Hence, we employ Bayesian technique to solve our modeled game. We consider a two-player static Bayesian game.

Figure 6.1 illustrates the extensive form of the Bayesian game. Here, node *Nature* determines the type of adversary. The objective of both players is to maximize their expected payoffs. This implies that we assume both players to be rational. This assumption is a generic assumption for a well-defined adversary-defender game [85]. In the defined game, the adversary plays a Bayesian strategy in order to minimize his chances of being detected, and the defender plays a Bayesian strategy in order to maximize his chance of responding to attacks without introducing high cost to the SPS system.

6.6.1 Bayesian Nash Equilibrium (BNE) Analysis

We analyze BNE based on the assumption that the type of adversary is unknown to the defense system. We assume that μ is a common prior, meaning that the adversary a knows the defender's belief of μ. Obviously, the adversary has private information behind its intention to target G_1 or G_2. In the following, we analyze the four possible pure-strategy BNEs that could exist.

1. If the adversary plays the pure strategy pair (**AT1 if type 1, AT1 if type 2**), then the expected payoff of the defense system playing the pure strategy CM1 is

$$E_d(CM1) = \mu(S_{G1} - C_{CM1}) + (1-\mu)(S_{G1} - C_{CM1}). \tag{6.7}$$

The expected payoff of the defense system playing pure strategy CM2 is

$$E_d(CM2) = \mu(-S_{G1} - C_{CM2}) + (1-\mu)(-S_{G1} - C_{CM2}). \tag{6.8}$$

So if $E_d(CM1) > E_d(CM2)$ or if $S_{G1} > \frac{C_{CM1}-C_{CM2}}{2}$ (which is always true since the assumption is $S_{G1} > C_{CM1}, C_{CM2}$), then the best response of the defense system is to play CM1. However, if the defense system plays CM1, then AT1 is not the best response when $-M_{AT1}^{T1} - C_{AT1} \leq -M_{AT2}^{T1} - C_{AT2}$ or $M_{AT1}^{T2} - C_{AT1} \leq M_{AT2}^{T2} - C_{AT2}$, and the adversary will move on to play AT2 instead. Therefore, in this case, (AT1 if type 1, AT1 if type 2, CM1) is not a BNE. However, if $-M_{AT1}^{T1} - C_{AT1} > -M_{AT2}^{T1} - C_{AT2}$ and $M_{AT1}^{T2} - C_{AT1} > M_{AT2}^{T2} - C_{AT2}$, the best response for the defender is CM1, and thus (AT1 if type 1, AT1 if type 2, CM1) is a pure-strategy BNE.

2. If the adversary plays the pure strategy pair (**AT1 if type 1, AT2 if type 2**), then the expected payoff for the defense system playing the pure strategy CM1 is

$$E_d(CM1) = \mu(S_{G1} - C_{CM1}) + (1-\mu)(-S_{G2} - C_{CM1}). \tag{6.9}$$

Similarly, the expected payoff of the defense system playing the pure strategy CM2 is

$$E_d(CM2) = \mu(-S_{G1} - C_{CM2}) + (1-\mu)(S_{G2} - C_{CM2}). \tag{6.10}$$

So if $E_d(CM1) > E_d(CM2)$ or if $\mu > \frac{2S_{G2}-C_{CM2}+C_{CM1}}{2S_{G1}+2S_{G2}}$, then the best response of the defense system is to play CM1. However, if the defense system plays CM1, then AT1 is not the best response for targeting G_1 when $-M_{AT1}^{T1} - C_{AT1} \leq -M_{AT2}^{T1} - C_{AT2}$, and it switches to AT2. If $M_{AT1}^{T2} - C_{AT1} \leq M_{AT2}^{T2} - C_{AT2}$, then AT2 is not the best response for targeting G_2, and the adversary moves on to play AT1. Hence, (AT1 if type 1, AT2 if type 2, CM1, μ) is not a pure-strategy BNE. If $\mu < \frac{2S_{G2}-C_{CM2}+C_{CM1}}{2S_{G1}+2S_{G2}}$, then the best response of the defense system is to play CM2. However, if $M_{AT2}^{T1} - C_{AT2} \geq M_{AT1}^{T1} - C_{AT1}$ then a type 1 adversary switches to AT2, and if $-M_{AT1}^{T1} - C_{AT1} \geq -M_{AT2}^{T2} - C_{AT2}$, then the type 2 adversary switches to AT1. Hence, (AT1 if type 1, AT2 if type 2, CM2, μ) is not a pure-strategy BNE.

Table 6.6: Case Based Analysis of Adversary-Type Uncertainty

	Type 1	**Type 1**	**Type 2**	**Type 2**		
	AT1	**AT2**	**AT1**	**AT2**	**CM1**	**CM2**
	(q_1)	$(1-q_1)$	(q_2)	$(1-q_2)$	(p)	$(1-p)$
Case 1	0.20	0.80	0.00	1.00	0.78	0.22
Case 2	1.00	0.00	0.80	0.20	0.22	0.78

3. If the adversary plays the pure strategy pair (**AT2 if type 1, AT1 if type 2**), then the expected payoff of the defense system playing the pure strategy CM1 is

$$E_d(CM1) = \mu(-S_{G2} - C_{CM1}) + (1-\mu)(S_{G1} - C_{CM1}). \tag{6.11}$$

The expected payoff of the defense system playing pure strategy CM2 is

$$E_d(CM2) = \mu(S_{G2} - C_{CM2}) + (1-\mu)(-S_{G1} - C_{CM2}). \tag{6.12}$$

So if $E_d(CM1) > E_d(CM2)$, or if $\mu < \frac{2S_{G1}+C_{CM2}-C_{CM1}}{2S_{G1}+2S_{G2}}$, then the best response by the defense system is to play CM1. As with the previous case, we can show that (AT2 if type 1, AT1 if type 2, CM1, μ) and (AT2 if type 1, AT1 if type 2, CM2, μ) are not pure-strategy BNEs.

4. If an adversary plays the pure strategy pair (**AT2 if type 1, AT2 if type 2**), then the expected payoff of the defense system playing the pure strategy CM1 is

$$E_d(CM1) = \mu(-S_{G2} - C_{CM1}) + (1-\mu)(-S_{G2} - C_{CM1}). \tag{6.13}$$

The expected payoff of the defense system playing pure strategy CM2 is

$$E_d(CM2) = \mu(S_{G2} - C_{CM2}) + (1-\mu)(S_{G2} - C_{CM2}). \tag{6.14}$$

So if $E_d(CM2) > E_d(CM1)$, or if $S_{G2} > \frac{C_{CM2}-C_{CM1}}{2}$ (which is always true since the assumption is that $S_{G2} > C_{CM1}, C_{CM2}$), then the best response of the defense system is to play CM2. However, if the defense system plays CM2, then AT2 is not the best attack strategy when $M^{T1}_{AT2} - C_{AT2} \leq M^{T1}_{AT1} - C_{AT1}$ and $-M^{T2}_{AT2} - C_{AT2} \leq -M^{T2}_{AT1} - C_{AT1}$. Hence, the adversary moves on to play AT1 instead. Therefore,

(AT2 if type 1, AT2 if type 2, CM2) is not a BNE. However, if $M^{T1}_{AT2} - C_{AT2} > M^{T1}_{AT1} - C_{AT1}$ and $-M^{T2}_{AT2} - C_{AT2} > -M^{T2}_{AT1} - C_{AT1}$, then the best response for the defender is CM2, and thus (AT2 if type 1, AT2 if type 2, CM2) is a pure-strategy BNE.

We previously showed that pure-strategy BNE exists when (i) AT1 if type 1, AT1 if type 2, CM1, $-M^{T1}_{AT1} - C_{AT1} > -M^{T1}_{AT2} - C_{AT2}$ and $M^{T2}_{AT1} - C_{AT1} > M^{T2}_{AT2} - C_{AT2}$, and (ii) AT2 if type 1, AT2 if type 2, CM2, $M^{T1}_{AT2} - C_{AT2} > M^{T1}_{AT1} - C_{AT1}$ and $-M^{T2}_{AT2} - C_{AT2} > -M^{T2}_{AT1} - C_{AT1}$. Although there exist a number of BNEs for particular pure strategies that meet certain criteria, here, we seek to find a mixed-strategy BNE for cases that do not result in a pure-strategy BNE. For this we must introduce two new belief probabilities. Setting the expected value of playing strategies CM1 and CM2 equal to each other we get

$$\begin{aligned}
&\mu q_1(S_{G1} - C_{CM1}) + \mu(1-q_1)(-S_{G2} - C_{CM1}) + \\
&\quad (1-\mu)q_2(S_{G1} - C_{CM1}) + (1-\mu)(1-q_2)(-S_{G2} - C_{CM1}) = \\
&\quad\quad \mu q_1(-S_{G1} - C_{CM2}) + \mu(1-q_1)(S_{G2} - C_{CM2}) + \\
&\quad\quad (1-\mu)q_2(-S_{G1} - C_{CM2}) + (1-\mu)(1-q_2)(S_{G2} - C_{CM2}). \quad (6.15)
\end{aligned}$$

Thus, the strategy pair (q_1 if type 1 adversary, q_2 if type 2 adversary, μ) is a mixed-strategy BNE, if Equation 6.15 is satisfied. Consequently, neither of the players can improve their payoffs by changing strategies.

6.6.2 Case Based Analysis

Let us describe how the proposed Bayesian game model analyzes potential responses given uncertainty regarding adversarial intentions. Here, we consider a scenario where the probability of targeting G_1 and G_2 (a type 1 adversary and a type 2 adversary) are the same in the SPS system.

We analyzed the response of the SPS system in two cases in which the preference of the SPS system is to protect either: (i) the G_1 security goal or (ii) the G_2 security goal. These two cases are exemplified using the two sets of numerical inputs introduced in Sections 6.4 and 6.5. Consequently, cases 1 and 2 are evaluated. We solved these two games using GAMBIT software [65]. Table 6.6 shows the probability of each strategy for both players a and d. In the rest of this subsection, we discuss in detail the response of the SPS system in each case.

- **Case 1: Uncertain Adversary Type vs. SPS System with G_1 Preference**

  We exemplify a type 1 adversary in an SPS system where G_1 is preferred over G_2. The numerical example of this case is shown in Figure 6.2. This game does not admit any NE solution in pure strategies. However, a unique NE is numerically computed in mixed strategies and shown in Table 6.6. At the NE, the type 1 adversary issues AT1 with the probability of 0.20. A reason for this low probability is the discouraging effect of the SPS system's capability to correctly respond to the attack. The NE strategy of the defender is CM1, with the probability of 0.78, and CM2 with the probability of 0.22.

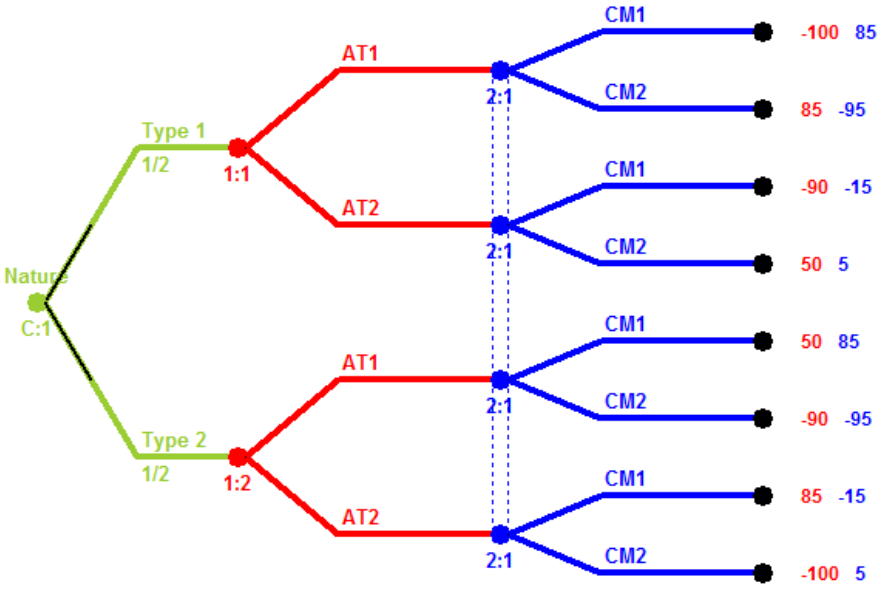


Figure 6.2: An Example of a Bayesian Security Game and Its Numerical Solution Obtained Using GAMBIT Software [65]

- **Case 2: Uncertain Adversary Type vs. SPS System with G_2 Preference**

  Here, we consider a scenario where the preference of the SPS system is to protect the G_2 security goal in the SPS system. The inputs are the same as in case 2 of the game model defined in Sections 6.4 and 6.5. At the NE, the probability of CM1 is 0.22, while the probability of CM2 is 0.78. The outcome of the game is based on consideration of G_2 preference in the SPS system.

In this section, we used Bayesian game technique to model and analyze the uncertainty about the type of adversary targeting the SPS system. In addition, we

provided a detailed analysis for various possible security goal preferences. In the next section, we describe the case study employing the proposed game-theoretic approach in selecting a proper countermeasure.

6.7 Realizing: Case Study of a Voice over IP

As a proof of concept for the proposed UBSP approach, we performed a case study with adversary-type uncertainty. The experiments are executed on a Voice over IP (VoIP) telephony system. The motivation for choosing such a system is the increasing number of application-layer flooding attacks in these systems. We study the case of application-layer attacks in which adversaries trigger various types of flooding attacks in an attempt to avoid detection. In this experimental evaluation, the following research questions are particularly taken into account:

• **RQ3.1 - Impact of adversary-type uncertainty**: What is the impact of considering uncertainty about adversary's type in the performance of the action selection in the SPS system?

• **RQ3.2 - Impact of security goal preference**: What is the impact of considering the preferences of various security goals in the performance of the action selection in the SPS system?

To answer the above questions, we conducted two types of flooding attacks while the SPS system is uncertain about the type of adversary. We analyzed and compared the impact of various mitigation approaches, including our proposed Bayesian game model.

Session Initiation Protocol (SIP) is a core protocol for real-time communication networks. VoIP communications rely on SIP protocol. Message flooding attacks exploit the common vulnerability of servers: their limited processing capability. Flooding attacks can be achieved with different SIP messages (REGISTER, INVITE, OPTIONS, etc.) [37]. In our experiments, we consider two types of flooding attacks: (i) REGISTER, and (ii) INVITE.

Protection of SIP servers against flooding attacks requires an online detection and mitigation technique to recognize such malicious requests and to drop them. Most research on the detection and mitigation of SIP flood attacks focuses on mining network-layer data [37]. However, this case study aims at incorporating the adversary-type uncertainty and the preferences of the SPS system when providing a response action.

6.7.1 Implementations

OpenJSIP [68] is an open source SIP service implemented in Java. The current version of the project, which we use in our experiments, is v0.0.4. OpenJSIP provides three services: *Proxy*, *Registrar*, and *Location Service*. The project is based on JainSIP [67], which is the standardized Java interface to the SIP for desktop and server applications.

To make a phone call, a user first needs to send a *registration request*, which will be directed to the Registrar server. If the request is handled successfully, then the user can make a *call request* to the Proxy server. In attempting to evolve OpenJSIP to an SPS system, our proposed decision-making approach is developed and integrated into OpenJSIP. The implemented adaptation manager monitors the requests and provides the necessary adaptation action upon receiving malicious requests. In the designed game model, the payoff values of actions are populated based on stakeholders' preferences. The payoff tables provide enough information for the decision-making engine to select proper actions.

Client traffic is generated using SIPp tool [117]. Three traffic generators are exploited, all running on the same machine with 3.00 GHz CPU and 4.00 GB memory. Two of the traffic generators represent legitimate users that try to call each other by sending REGISTER and INVITE requests to the Registrar and Proxy servers. The third traffic generator represents adversaries that send malicious traffic while the system is uncertain about the type of adversary. Two possible attacks studied in our experiments are (i) flooding the Registrar server with malicious REGISTER requests (AT1), and (ii) flooding the Proxy server with malicious INVITE requests (AT2).

6.7.2 Attack Scenario

The case study security goals are to sustain the availability of the *Registrar* and *Proxy* servers during an attack. Hence, the objective is to increase the number of successful registration and call requests that are initiated by legitimate users. Accordingly, legitimate users are successfully provided with their expected service. In the experiments, the security goals of the SPS system are (i) availability of the Registrar server (G_1), and (ii) availability of the Proxy server (G_2). Consequently, two type of adversaries threaten the SPS system: (i) a type 1 adversary which targets the availability of the Registrar server, and (ii) a type 2 adversary which targets the availability of the Proxy server. The SPS system can take two countermeasures: (i) dropping

the incoming REGISTER requests (CM1), and (ii) dropping the incoming INVITE requests (CM2).

Two attack scenarios have been designed. In both, the adversary changes its strategy in order to confuse the SPS system by sending both malicious REGISTER and INVITE requests. In the first attack scenario, the intention of the adversary is to target the availability of the Registrar server (a type 1 adversary). In the second attack scenario, the intention of the adversary is to target the availability of the Proxy server (a type 2 adversary).

In both attack scenarios, two traffic generators represent legitimate users try to call each other. First, both caller and callee send a REGISTER request to register their IDs at the Registrar server. Then, each caller tries to call a predefined callee by sending an INVITE request to the Proxy server. The server looks for the callee, and if the callee is available, it informs the caller to establish the call. The caller holds the line for 1 second and then tries to terminate the call by sending a BYE message to the server. Both sides terminate the call when they receive an ACK from the server. The clients keep generating traffic for a total of 1 minute.

Prior to starting the test, we ran a stress test to discover the saturation point of the server on the running computer. We tested the system with increasing traffic and monitored at which point the system failed to respond to client requests. The observed limit was around 40 Calls Per Second (CPS). This rate was used to flood the Registrar and Proxy servers with malicious requests while the adversary changed its strategy by changing the type of flooding attack. The attack starts after 5 seconds, when server and both clients start to run. The attack continues for the rest of the minute or until the system breaks down. The experimental setup satisfactorily illustrates the performance of the proposed approach when facing uncertainty about the type of adversary. The following subsection presents the outcome of the experiments.

6.8 Obtained Results

The experiments in this section evaluate the mitigation of attacks in the case of adversary-type uncertainty. Six approaches are considered for action selection by the defense system:

1. **No Adaptation**: selects no mitigation action during the attack.

2. **Protecting** G_1: selects countermeasures solely to protect G_1, which means dropping malicious REGISTER requests without considering the uncertainty about the strategy and intention of adversaries.

3. **Protecting** G_2: selects countermeasures solely to protect G_2, which means dropping malicious INVITE requests without considering the uncertainty about the strategy and intention of adversaries.

4. **Known Adversary-Type**: selects countermeasures assuming that the intention behind the attack is known to the system and it is considered in the action selection. (Game model in Section 6.4 and Section 6.5)

5. **Adversary-Type Uncertainty Case 1**: selects countermeasure using Bayesian game model while the SPS system has higher preference for G_1.

6. **Adversary-Type Uncertainty Case 2**: selects countermeasure using Bayesian game model while the SPS system has higher preference for G_2.

In the following, we aim at providing the relevant answers for each of the two research questions introduced in the beginning of this section. To make this possible, we need to measure the effectiveness of all six approaches. An appropriate method is to assess the percentage of successful registrations and successful calls for legitimate users during type 1 and type 2 adversary attacks.

Table 6.7 summarizes the result of the first attack scenario, whose intention was to target the availability of the Registrar server. In this case, *Protecting* G_1, *Known Adversary Type*, and *Adversary-Type Uncertainty Case 1* are among the successful approaches. However, *Protecting* G_1 performs best solely in terms of successful registrations. *Known Adversary Type* technique is based on the assumption that the SPS system is aware of the intention and strategy of adversaries, while in *Adversary-Type Uncertainty Case 1*, the SPS system is uncertain about the type of adversary. The results are comparable in both techniques in percentage of successful registrations (addressing *RQ3.1*). In terms of successful calls, *Adversary-Type Uncertainty Case 1* has less value compared to *Adversary-Type Uncertainty Case 2* because in the former the security goal G_1 had higher priority. Hence, the SPS system selected countermeasures that protected this goal (addressing *RQ3.2*).

Table 6.8 summarizes the results of the second attack scenario, whose intention was to target the availability of the Proxy server. As with the previous attack scenario, it appears that there is no significant difference between *Protecting* G_2, *Known Adversary Type*, and *Adversary-Type Uncertainty Case 2* techniques. The

Table 6.7: Type 1 Adversary: Targeting the Availability of Registrar Server (G_1)

	No Adaptation	Protecting G_1	Protecting G_2	Known Adversary-Type	Adversary-Type Uncertainty Case 1	Adversary-Type Uncertainty Case 2
% Suc. Reg.	80	97	78	95	**95**	86
% Suc. Call	49	70	48	88	**83**	85

Table 6.8: Type 2 Adversary: Targeting the Availability of Proxy Server (G_2)

	No Adaptation	Protecting G_1	Protecting G_2	Known Adversary-Type	Adversary-Type Uncertainty Case 1	Adversary-Type Uncertainty Case 2
% Suc. Reg.	44	52	100	92	86	**99**
% Suc. Call	27	30	100	93	61	**95**

response to *RQ3.1*, based on the results, is that our proposed approach to address uncertainty obtains results comparable to no uncertainty in the type of adversary (*Known Adversary Type*). *Protecting* G_2 is the most effective approach as it aims at protecting the Proxy server, which is reflected during countermeasure selection. However, this approach assumes that the intention of the adversary is known to the SPS system.

Comparison of *Adversary-Type Uncertainty Case 1* and *Adversary-Type Uncertainty Case 2* approaches, indicates that including the preferences of the security goals is more satisfactory when the adversary targets the security goal that is of higher preference. If the adversary targets the availability of the Registrar server (G_1), then the Case 1 uncertainty approach results in more-successful registrations and calls. If the adversary targets the availability of the Proxy server (G_2), then the Case 2 obtains more effective results than Case 1 (addressing *RQ3.2*).

Our case study illustrates that the proposed Bayesian game theoretic approach effectively addresses uncertainty about adversary type by selecting a countermeasure that maximizes the expected payoff of the SPS system considering the strategy of the adversary and the security goals of the SPS system.

6.9 Summary

This chapter extends the proposed framework by providing a decision-making model that addresses adversary-type uncertainty. For this purpose, this work deals with how

to keep security goals in the SPS system and to involve the intention and strategy of an adversary in the decision model. In our UBSP approach, we used Bayesian game theoretic technique to analyze the battle between the adversary and defense system. The defense system forms beliefs and measures uncertainty to evaluate the type of opponent and selects a countermeasure according to the adversarial strategy. The adversary keeps evaluating the cost and benefit of initiating an attack and selects a strategy aiming to minimize punishment. We provide Bayesian Nash Equilibrium (BNE) analysis of the game, along with a case study. The experiments conducted on a Java-based VoIP telephony system, reveal that the proposed UBSP approach efficiently selects proper countermeasures in the case of adversary-type uncertainty. The selected countermeasures satisfy the preferred security goals while considering the benefit and the cost of applying countermeasures.

Chapter 7

Concluding Remarks and Future Directions

This thesis investigates the process of engineering a decision-making engine in a self-protecting software system. This chapter summarizes the findings of this thesis and presents future directions. Section 7.1 presents the contributions of the thesis, Section 7.2 sums up the answers to the research questions, Section 7.3 discusses the future works that would extend this research, finally, Section 7.4, makes concluding remarks.

7.1 Contributions

The contributions of this thesis are stated in Chapter 1. This section discusses these contributions in the light of the material that is covered in Chapters 3-6. These contributions include:

- **a novel mapping from high-level quality/malicious goals to quantitative utility values**. The advantage of modeling and incorporating goals is discussed in our paper [44]. With the aid of the employed goal models such as SIG (in Chapter 4) and GAAM (in Chapter 5), the decision-making engine considers the trade-offs among quality goals of the software system. What distinguishes the modeling phase of the proposed framework from other modeling approaches is the mapping of the satisfaction level of the goals to quantitative values. For example in Chapter 5, MARGIN models the quality goals with

GAAM from high-level perspective of stakeholders down to low-level operational goals. The model is improved by mapping the satisfaction level of these goals to the reward values which are used to update the decision model.

By fusing the malicious goals of attackers, the intention behind the attack is modeled into the decision model which helps the self-protecting software system to deal with uncertain-type attacks. In Chapter 4, IBSP models malicious goals of attackers with the aid of SIG. The model is further extended to map the satisfaction level of these goals to quantitative utility values. In Chapter 6, the costs and benefits of the adversary to achieve its malicious goal is measured and incorporated in the probabilistic-based decision model of UBSP.

- **a unique modeling of the interdependencies among defense and attack strategies**. Self-protection is provided against different attacks by considering the interdependency among strategies. This information is modeled into the decision model by capturing the positive/negative impact of the strategies on the quality/malicious goals. This is a distinctive feature of the proposed framework. In Chapter 4, IBSP evaluates the interdependencies into the utility values. Accordingly, utility values are used to populate the players' payoffs in the stochastic game model.

- **a learning-based decision model during runtime**. The proposed framework is extended in Chapter 5 to capture the impact of the previous applied countermeasure and to update the decision model. The learning-base decision-making engine in Chapter 5, called MARGIN, demonstrates the ability to learn from the positive/negative impact of the taken countermeasure. MARGIN is distinguishable in terms of its capability to adjust itself to take effective countermeasure facing dynamic strategies of attackers.

- **an innovative decision model for adversary-type uncertainty**. Chapter 6 demonstrates that UBSP employs Bayesian game model to handle the ambiguity about the utilities according to the type of the attacker. What distinguishes UBSP is the capability to effectively addresses uncertainty about the adversary type by selecting a countermeasure that maximizes the expected payoff of the SPS system considering the strategy of the adversary and the security preferences of the SPS system.

- **a step-wise plug-and-play framework**. Plug-and-play in our framework is the ability to employ (i) different *goal models* depending on the properties of the available data that can be used in the model and (ii) various *game-theory*

techniques based on the security goal and requirements of the software system. In order to illustrate the *plug-and-play* capability of our proposed framework, a summary of the relations of each approach to the original framework is provided in Table 7.1. Each row represents a decision-making approach and briefly specifies three phases which are based on the phases in the proposed framework. The first approach, called IBSP, is a strategy-aware approach employing stochastic games (explained in Chapter 4). The second approach, called MARGIN, is a learning-based approach utilizing Markov games (discussed in Chapter 5). The third approach, called UBSP, is a probabilistic-based approach that takes into account the adversary-type uncertainty by applying Bayesian games (described in Chapter 6). These three approaches exhibit the plug-and-play aspect of the proposed framework. For each of these approaches, we describe the rational behind the choice of the game-theoretic technique in Sections 4.4, 5.4, and 6.6 where the designing phase is described.

Table 7.1: Publication linked with Research Methodology

Phase / **Publication**	**Modeling Goals**	**Designing Game-Theoretic Mechanism**	**Realizing Adaptation Manager**
Mitigating Dynamic Attacks Using Multi-Agent Game-Theoretic Techniques [44]	Soft-goal Interdependency Graph (SIG)	Multi-Agent Game-Theoretic Technique (Strategy-aware approach)	Case study example considering the interdependencies of strategies
Strategy-Aware Mitigation Using Markov Games for Dynamic Application-Layer Attacks [45]	Goal-driven models such as Goal-Action-Attribute Model (GAAM)	Markov Game Technique (Learning-based approach)	MATLAB Simulink - simulation of a web application
A Bayesian Game Decision-Making Model for Uncertain Adversary Types [46]	Using NFR to model cost & benefits of attacks /countermeasures	Bayesian Game Technique (Considers uncertainty)	VoIP Java-based telephony system

Each of the IBSP, MARGIN, or UBSP approaches (described in Chapters 4, 5, and 6) addresses a particular research challenge in adaptive application security.

Therefore, they are not comparable with each other. For instance, IBSP tackles fusing the attacker while UBSP addresses adversary-type uncertainty. MARGIN is a closed loop approach to incorporate feedback from the previous countermeasures taken while IBSP and UBSP are open loop approaches.

7.2 A Summary of Research Questions

This section revisits the research questions presented in this thesis and provides short answers based on the findings presented in the different chapters.

- **RQ1: How should the framework incorporate the incentives of the attacker?**

  Chapter 4 of this thesis proposes a mitigation technique (IBSP) that aims at incorporating the strategy of the attacker and hence the inherent interdependencies between the software system and the attacker into the decision model.

  • *Modeling Goals*: Goals for each player can be modeled by the aid of Soft-goal Interdependancy Graph (SIG) [29]. SIG has notations that can represent the magnitude of the impact of strategies on each goal.

  • *Designing Game-Theoretic Mechanism*: Using the modeled goals in SIG and relating them to the strategies of the system and the attacker, a game theoretic approach can be employed to find the equilibrium.

  • *Realizing Adaptation Manager*: A case study example is provided to illustrate the advantage of considering the interdependencies of strategies while applying a countermeasure. We provided a formal analysis of the proposed modeling and designing technique with the aid of model checking of Stochastic Multiplayer Games (SMGs) [23] using PRISM-games [24].

- **RQ2: How should the framework learn from previous action selections?**

  Chapter 5 of this thesis proposes a learning-based mitigation technique (MARGIN) which is based on Markov game technique. Markov game employs reinforcement learning, hence the proposed technique can be adaptive to changes in the strategy of the attacker (dynamic attacks).

  • *Modeling Goals*: Goal-driven models such as Goal-Action-Attribute Model (GAAM) [110] can be employed to formulate the high-level quality goals of the system to actions.

• *Designing Game-Theoretic Mechanism*: Markov games, also known as stochastic games, are the extension of the single agent Markov Decision Process (MDP) to the multi-agent case. Markov games add an extra concept to game theory which is a set of system states.

• *Realizing Adaptation Manager*: As a proof of concept, this research conducts a study on a case of dynamic application-layer denial of service attacks. The simulation results demonstrate that our approach performs effectively while encountering different attack strategies.

- **RQ3: How should the framework support adversary-type uncertainty?**

  Chapter 6 of this thesis proposes an uncertainty-based decision-making approach (UBSP) to address uncertain adversary types. Our game-theoretic model formalizes possible intentions of adversaries along with the security preferences of the software system.

  • *Modeling Goals*: Non-Functional Requirement (NFR) modeling techniques are used to model costs and benefits of attacks/countermeasures.

  • *Designing Game-Theoretic Mechanism*: Our Bayesian game-theoretic model formalizes possible intentions of adversaries along with the security preferences of the software system.

  • *Realizing Adaptation Manager*: As a proof of concept for the proposed mitigation technique, we performed a case study with adversary-type uncertainty. The experiments are executed on a java-based Voice over IP (VoIP) telephony system. The results illustrate that the proposed technique effectively addresses uncertainty about adversary type by selecting a countermeasure that maximizes the expected payoffs of the SPS system considering the strategy of the adversary and the security goal preferences of the SPS system.

7.3 Future Work

The proposed plug-and-play decision-making framework laid the groundwork for several future directions. The following is a list of the possible future works:

- extending the game-theoretic decision-making engine in the framework to support a course of actions. As well-panned multi-stage attacks are becoming

more common in application-layer attacks, the goal is to add the capability of countering such attacks. For attack types that are consist of multiple steps to fulfill the attacker's goal, the progress of the attack can be modeled into the decision engine. For example, if an attack is at its advance stage then the defense system need to apply a more effective countermeasure even though it is costly.

- extending the learning of the decision model to not only learn the type of known attacks, but also to be capable of learning unknown (zero-day) attacks. A software system always has unknown vulnerabilities that are yet to be discovered. Hence, a software system is threatened by zero-day attacks. One of the extensions of the decision model could relating system/user attributes to known attack as well as relating those attributes to possible unknown attacks. This requires to define additional formulas in the decision model that capture (i) the relation between an unknown attack and its impact on the system/user attributes, and (ii) the uncertainty about the maliciousness of the detected behavior.

- supporting a wider range of game-theoretic approaches. In this thesis, the plug-and-play framework supports three various game-theoretic techniques. In the future, I will work on more game-theoretic approaches which are capable of addressing different security requirements of the software system. We plan to employ techniques from repeated games and fuzzy games to learn the best strategy when facing unknown types of attacks.

- extending the framework to support security in cyber-physical systems. Cyber-physical systems are distributed, software-intensive systems that control tightly integrated and networked computational and physical components [103]. A key challenge in these systems is the continuous assurance of quality goals at runtime. Traditionally, satisfaction of quality goals is gained through a variety of requirement engineering and analysis at the design and development time such as security requirement engineering [39]. Since, cyber-physical systems are at core adaptive, the verification for the satisfaction of the quality goals (such as security) need to be performed at runtime. New security challenges have emerged with the exponential growth of cyber-physical systems [105] and various threats and attacks have been introduced [63]. In the future, I am planing to support coordinating different defense strategies that tackle various types of attacks in cyber-physical systems.

- further expansion of the framework to support adding/removing strategies during runtime. This capability will be beneficial to support self-protection of various software systems which are working together. A major trend in software engineering is called Internet of Things (IoT). This research area has introduced challenges in security and specifically collaboration of strategies [22]. When a large number of unsecured devices are connected to the internet, the decision-making engine requires to add/remove strategies on the fly based on the connected devices. I am planning to enhance the framework to support updating: (i) the defined actions in the goal model, and (ii) the defined strategies in the game. Such runtime capability will facilitate deploying our game-theoretic framework on the IoT and update it based on the edge devices getting connected/disconnected.

7.4 Conclusion

With the advent of cyber-physical systems and Internet-of-Things, software security is facing new challenges. A software system, in such complex and heterogeneous environment, requires to be equipped with advanced self-protection techniques. This thesis has paved the road to enable intelligent decision-making in self-protecting software systems. The proposed plug-and-play framework is explored in three various directions (incentive-based, learning-based, and probabilistic-based) depending on the security requirements and characteristics of the attacks that the software system is facing as well as the characteristics of attackers. Accordingly, a game-theoretic technique is employed considering the modeled knowledge of the software system and attack scenarios. In these three extensions, the framework is successfully realized in various self-protecting software systems. The three extensions exhibit the plug-and-play capability of the framework.

References

[1] ISO/IEC 25010:2011. [Online.] Available: http://www.iso.org/iso/cataloguedetail.htm?csnumber=35733. [Accessed: 7- January- 2018].

[2] Ian Alexander. Misuse cases: Use cases with hostile intent. *IEEE Software*, 20(1):58–66, 2003.

[3] André Almeida, Nelly Bencomo, Thais Batista, Everton Cavalcante, and Francisco Dantas. Dynamic decision-making based on nfr for managing software variability and configuration selection. In *Proceedings of the Annual ACM Symposium on Applied Computing*, pages 1376–1382, 2015.

[4] Tansu Alpcan and Tamer Başar. *Network security: A decision and game-theoretic approach*. Cambridge University Press, 2010.

[5] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[6] Daniel Amyot, Sepideh Ghanavati, Jennifer Horkoff, Gunter Mussbacher, Liam Peyton, and Eric Yu. Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 25(8):841–877, 2010.

[7] Christopher Bailey, Lionel Montrieux, Rogério de Lemos, Yijun Yu, and Michel Wermelinger. Run-time generation, transformation, and verification of access control models for self-protection. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 135–144, 2014.

[8] Cornel Barna, Mark Shtern, Michael Smit, Vassilios Tzerpos, and Marin Litoiu. Mitigating DoS attacks using performance model-driven adaptive algorithms. *ACM Transactions on Autonomous and Adaptive Systems*, 9(1):3:1–3:26, 2014.

[9] Nasim Beigi-Mohammadi, Cornel Barna, Mark Shtern, Hamzeh Khazaei, and Marin Litoiu. CAAMP: Completely automated ddos attack mitigation platform in hybrid clouds. In *Proceedings of the International Conference on Network and Service Management*, pages 136–143, 2016.

[10] Nelly Bencomo, Amel Belaggoun, and Valerie Issarny. Dynamic decision networks for decision-making in self-adaptive systems: a case study. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 113–122, 2013.

[11] Jonathan J Blount, Daniel R Tauritz, and Samuel A Mulder. Adaptive rule-based malware detection employing learning classifier systems: a proof of concept. In *Proceedings of Annual Computer Software and Applications Conference Workshops*, pages 110–115, 2011.

[12] Volha Bryl, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Designing security requirements models through planning. In *Proceedings of International Conference on Advanced Information Systems Engineering*, pages 33–47, 2006.

[13] Javier Cámara, David Garlan, Won Gu Kang, Wenxin Peng, and Bradley Schmerl. Uncertainty in self-adaptive systems. Technical Report CMU-ISR-17-110, Institute for Software Research, Carnegie Mellon University, July 2017.

[14] Javier Cámara, David Garlan, Gabriel A. Moreno, and Bradley Schmerl. Evaluating trade-offs of human involvement in self-adaptive systems. Technical report, 2016.

[15] Javier Cámara, David Garlan, Gabriel A. Moreno, and Bradley Schmerl. *Analyzing Self-Adaptation via Model Checking of Stochastic Games*. Number 9640. Springer, 2017.

[16] Javier Cámara, Gabriel A. Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 155–164, 2014.

[17] Javier Cámara, Gabriel A Moreno, and David Garlan. Reasoning about human participation in self-adaptive systems. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 146–156, 2015.

[18] Javier Cámara, Gabriel A Moreno, David Garlan, and Bradley Schmerl. Analyzing latency-aware self-adaptation using stochastic games and simulations. *ACM Transactions on Autonomous and Adaptive Systems*, 10(4):23, 2016.

[19] Javier Cámara, Wenxin Peng, David Garlan, and Bradley Schmerl. Reasoning about sensing uncertainty in decision-making for self-adaptation. In *Proceedings of the International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems*, 2017.

[20] Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Taxonomy of slow DoS attacks to web applications. In *Recent Trends in Computer Networks and Distributed Systems Security*, volume 335 of *Communications in Computer and Information Science*, pages 195–204. Springer Berlin Heidelberg, 2012.

[21] Dawn M Cappelli, Andrew P Moore, and Randall F Trzeciak. *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012.

[22] Yassine Chahid, Mohamed Benabdellah, and Abdelmalek Azizi. Internet of things security. In *Proceedings of the International Conference on Wireless Technologies, Embedded and Intelligent Systems*, pages 1–6, 2017.

[23] Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[24] Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Prism-games: A model checker for stochastic multi-player games. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 185–191, 2013.

[25] Taolue Chen and Jian Lu. Probabilistic alternating-time temporal logic and model checking algorithm. In *Proceedings of International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 35–39, 2007.

[26] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009.

[27] Betty H.C. Cheng, RogÃľrio Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina

Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems, Lecture Notes in Computer Science*, volume 5525, pages 1–26. Springer, 2009.

[28] Keywhan Chung, Charles A Kamhoua, Kevin A Kwiat, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. Game theory with learning for cyber security monitoring. In *Proceedings of the International Symposium on High Assurance Systems Engineering*, pages 1–8, 2016.

[29] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-functional Requirements in Software Engineering*, volume 5. The Kluwer International Series in Software Engineering, 2000.

[30] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[31] Benoit Claudel, Noël De Palma, Renaud Lachaize, and Daniel Hagimont. Self-protection for distributed component-based applications. In *Proceedings of International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 184–198, 2006.

[32] IBM Co. Autonomic computing 8 elements. [Online.] Available: http://www.research.ibm.com/autonomic/overview/elements.html. [Accessed: 7-January- 2018].

[33] IBM Co. Towards Autonomic Networking Middleware - IBM Research. [Online.] Available: http://www.research.ibm.com/people/a/akeller/Data/ngnm2005_slides.pdf/. [Accessed: 7- January- 2018].

[34] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[35] Morton D. Davis. *Game theory: a nontechnical introduction*. Dover Publications, 1997.

[36] Cuong T Do, Nguyen H Tran, Choongseon Hong, Charles A Kamhoua, Kevin A Kwiat, Erik Blasch, Shaolei Ren, Niki Pissinou, and Sundaraja Sitharama Iyengar. Game theory for cyber security and privacy. *ACM Computing Surveys*, 50(2):30, 2017.

[37] Sven Ehlert, Dimitris Geneiatakis, and Thomas Magedanz. Survey of network security systems to counter SIP-based denial-of-service attacks. *Computers & Security*, 29(2):225–243, 2010.

[38] Golnaz Elahi and Eric Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *Proceedings of Conceptual Modeling-ER*, pages 375–390, 2007.

[39] Golnaz Elahi, Eric Yu, and Nicola Zannone. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, 2010.

[40] Ahmed Elkhodary and Jon Whittle. A survey of approaches to adaptive application security. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 20–26, 2007.

[41] Mahsa Emami-Taba. Decision-making in self-protecting software systems: a game-theoretic approach. In *Proceedings of the International Conference on Software Engineering Companion*, pages 77–79, 2017.

[42] Mahsa Emami-Taba. A game-theoretic decision-making framework for engineering self-protecting software systems. In *Proceedings of the International Conference on Software Engineering Companion*, pages 449–452, 2017.

[43] Mahsa Emami-Taba, Mehdi Amoui, and Ladan Tahvildari. On the road to holistic decision making in adaptive security. *Technology Innovation Management Review*, 3(8):59–64, 2013.

[44] Mahsa Emami-Taba, Mehdi Amoui, and Ladan Tahvildari. Mitigating dynamic attacks using multi-agent game-theoretic techniques. In *Proceedings of IBM International Conference on Computer Science and Software Engineering*, pages 375–378, 2014.

[45] Mahsa Emami-Taba, Mehdi Amoui, and Ladan Tahvildari. Strategy-aware mitigation using Markov games for dynamic application-layer attacks. In *Proceedings of International Symposium on High-Assurance Systems Engineering*, pages 134–141, 2015.

[46] Mahsa Emami-Taba and Ladan Tahvildari. A Bayesian game decision-making model for uncertain adversary types. In *Proceedings of IBM International Conference on Computer Science and Software Engineering*, pages 39–49, 2016.

[47] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. Taming uncertainty in self-adaptive software. In *Proceedings of the European conference on Foundations of software engineering*, pages 234–244, 2011.

[48] Antti Evesti and Eila Ovaska. Comparison of adaptive information security approaches. *ISRN Artificial Intelligence*, pages 1–18, 2013.

[49] Andrew Fielder, Emmanouil Panaousis, Pasquale Malacaria, Chris Hankin, and Fabrizio Smeraldi. Game theory meets information security management. In *Proceedings of Information Security Conference*, pages 15–29, 2014.

[50] Dominik Fisch, Martin Jänicke, Edgar Kalkowski, and Bernhard Sick. Techniques for knowledge acquisition in dynamically changing environments. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1):16:1–16:25, 2012.

[51] János Fülöp. Introduction to decision making methods. In *Proceedings of Biodiversity and Ecosystem Informatics Workshop*, 2005, 15 pages.

[52] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[53] Andrey Garnaev, Melike Baykal-Gursoy, and H Vincent Poor. Incorporating attack-type uncertainty into network protection. *IEEE Transactions on Information Forensics and Security*, 9(8):1278–1287, 2014.

[54] Holger Giese, Nelly Bencomo, Liliana Pasquale, Andres J Ramirez, Paola Inverardi, Sebastian Wätzoldt, and Siobhán Clarke. Living with uncertainty in the age of runtime models. In *Models@ run.time : Foundations, Applications, and Roadmaps*, pages 47–100. Springer, 2014.

[55] Paolo Giorgini, John Mylopoulos, and Roberto Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.

[56] Cleotilde Gonzalez, Noam Ben-Asher, and Don Morrison. Dynamics of decision making in cyber defense: Using multi-agent cognitive modeling to understand cyberwar. In *Proceedings of the Theory and Models for Cyber Situation Awareness*, pages 113–127. 2017.

[57] Assane Gueye. *A game theoretical approach to communication security*. PhD thesis, University of California, Berkeley, 2011.

[58] Anna V Guglielmi and Leonardo Badia. Analysis of strategic security through game theory for mobile social networks. In *Proceedings of Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, pages 1–6, 2017.

[59] Charles B. Haley, Robin Laney, Jonathan D. Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.

[60] Sara Hassan, Nelly Bencomo, and Rami Bahsoon. Minimizing nasty surprises with better informed decision-making in self-adaptive systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 134–144, 2015.

[61] Jennifer Horkoff and Eric Yu. A qualitative, interactive evaluation procedure for goal-and agent-oriented models. In *Proceedings of CEUR workshop in CAiSE Forum*, 2009.

[62] Jennifer Horkoff and Eric Yu. Analyzing goal models: different approaches and how to choose among them. In *Proceedings of the ACM Symposium on Applied Computing*, pages 675–682, 2011.

[63] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security–a survey. *IEEE Internet of Things Journal*, PP(99), 2017.

[64] IBM. An architectural blueprint for autonomic computing. Available: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf. *IBM White Paper*, 2006.

[65] Gambit Inc. Gambit Game Theory Analysis Software and Tools. [Online.] Available: http://gambit.sourceforge.net/. [Accessed: 7- January- 2018].

[66] MathWorks Inc. Matlab simevents toolbox. [Online.] Available: http://www.mathworks.com/products/simevents/. [Accessed: 7- January- 2018].

[67] JAIN SIP. The Standardized Java Interface to the Session Initiation Protocol. [Online.] Available: https://jsip.java.net/, [Accessed: 7- January- 2018].

[68] Open Java SIP. [Online.] Available: https://code.google.com/p/openjsip/ [Accessed: 7- January- 2018].

[69] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *Proceedings of Symposium on Networked Systems Design & Implementation*, pages 287–300, 2005.

[70] Ralph L. Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value tradeoffs*. John Wiley, 1976.

[71] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[72] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[73] Robert Laddaga. Self-adaptive software. Technical report, DARPA BAA. 98–12, 1997.

[74] Hemank Lamba, Thomas J. Glazier, Javier Cámara, Bradley Schmerl, David Garlan, and Jürgen Pfeffer. Model-based cluster analysis for identifying suspicious activity sequences in software. In *Proceedings of the International Workshop on Security and Privacy Analytics*, 2017.

[75] Seokcheol Lee, Sungjin Kim, Ken Choi, and Taeshik Shon. Game theory-based security vulnerability quantification for social internet of things. *Future Generation Computer Systems*, 2017.

[76] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. A data mining framework for building intrusion detection models. In *Proceedings of Symposium on Security and Privacy*, pages 120–132, 1999.

[77] RogÃĬrio Lemos et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science*, volume 7475, pages 1–32. Springer, 2013.

[78] Kevin Leyton-Brown and Yoav Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan and Claypool Publishers, 2008.

[79] Feng Li and Jie Wu. Hit and run: a bayesian game between malicious and regular nodes in MANETs. In *Proceedings of Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 432–440, 2008.

[80] Tong Li, Jennifer Horkoff, and John Mylopoulos. Analyzing and enforcing security mechanisms on requirements specifications. In *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 115–131, 2015.

[81] Xiannuan Liang and Yang Xiao. Game theory for network security. *IEEE Communications Surveys and Tutorials*, 15(1):472–486, 2013.

[82] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of International Conference on Machine Learning*, pages 157–163, 1994.

[83] KJ Ray Liu and Beibei Wang. *Cognitive radio networking and security: A game-theoretic view*. Cambridge University Press, 2010.

[84] Peng Liu, Wanyu Zang, and Meng Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Transactions on Information and System Security*, 8(1):78–118, 2005.

[85] Yu Liu, Cristina Comaniciu, and Hong Man. A bayesian game approach for intrusion detection in wireless ad hoc networks. In *Proceedings of workshop on Game theory for communications and networks*, 2006.

[86] Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D Santambrogio, Anant Agarwal, and Alberto Leva. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Transactions on Autonomous and Adaptive System*, 7(4):36, 2012.

[87] Hoda Maleki, Saeed Valizadeh, William Koch, Azer Bestavros, and Marten van Dijk. Markov modeling of moving target defense games. In *Proceedings of the Workshop on Moving Target Defense*, pages 81–92, 2016.

[88] Mohammadhossein Manshaei, Quanyan Zhu, Tansu Alpcan, Tamer Başar, and Jean-Pierre Hubaux. Game theory meets network security and privacy. *ACM Computing Surveys*, 45(3):1–25, 2013.

[89] Bill McCarty. Botnets: Big and bigger. *IEEE Security & Privacy*, 1(4):87–90, 2003.

[90] Gary McGraw. *Software security: building security in*. Addison-Wesley Professional, 2006.

[91] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[92] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In *Proceedings of the International Conference on Autonomic Computing*, pages 147–156, 2016.

[93] Haralambos Mouratidis. Secure software systems engineering: the secure tropos approach. *Journal of Software*, 6(3):331–339, 2011.

[94] Thanh H Nguyen, Francesco M Delle Fave, Debarun Kar, Aravind S Lakshminarayanan, Amulya Yadav, Milind Tambe, Noa Agmon, Andrew J Plumptre, Margaret Driciru, Fred Wanyama, et al. Making the most of our regrets: Regret-based solutions to handle payoff uncertainty and elicitation in green security games. In *Proceedings of Conference on Decision and Game Theory for Security*, pages 170–191, 2015.

[95] Thanh H Nguyen, Arunesh Sinha, and Milind Tambe. Conquering adversary behavioral uncertainty in security games: An efficient modeling robust based algorithm. *AAAI (Student Abstract)*, 2016.

[96] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[97] Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini. Modelling and reasoning about security requirements in socio-technical systems. *Data & Knowledge Engineering*, 98:123–143, 2015.

[98] Ashutosh Pandey, Gabriel A. Moreno, Javier Cámara, and David Garlan. Hybrid planning for decision making in self-adaptive systems. In *Proceedings of the 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2016.

[99] Liliana Pasquale, Claudio Menghi, Mazeiar Salehie, Luca Cavallaro, Inah Omoronyia, and Bashar Nuseibeh. SecuriTAS: A tool for engineering adaptive security. In *Proceedings of International Symposium on the Foundations of Software Engineering*, pages 19:1–19:4, 2012.

[100] Luis H Garcia Paucar and Nelly Bencomo. Re-pref: Support for reassessment of preferences of non-functional requirements for better decision-making in self-adaptive systems. In *Proceedings of the IEEE International Requirements Engineering Conference*, pages 411–414, 2016.

[101] Luis Hernan Garcia Paucar, Nelly Bencomo, and Kevin Kam Fung Yuen. Juggling preferences in a world of uncertainty. In *Proceedings of the IEEE International Requirements Engineering Conferencel*, pages 430–435, 2017.

[102] Andres J Ramirez, Adam C Jensen, and Betty HC Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 99–108, 2012.

[103] Danda B Rawat, Joel JPC Rodrigues, and Ivan Stojmenovic. *Cyber-physical systems: from theory to practice*. CRC Press, 2015.

[104] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *Proceedings of International Conference on System Sciences*, pages 1–10, 2010.

[105] Ivan Ruchkin, Ashwini Rao, Dio De Niz, Sagar Chaki, and David Garlan. Eliminating inter-domain vulnerabilities in cyber-physical systems: An analysis contracts approach. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems Security and Privacy*, 2015.

[106] Mazeiar Salehie, Liliana Pasquale, Inah Omoronyia, Raian Ali, and Bashar Nuseibeh. Requirements-driven adaptive security: Protecting variable assets at runtime. In *Proceedings of International Requirements Engineering Conference*, pages 111–120, 2012.

[107] Mazeiar Salehie and Ladan Tahvildari. Autonomic computing: emerging trends and open problems. In *Proceedings of ICSE Workshop on Design and Evolution of Autonomic Application Software*, pages 82–88, 2005.

[108] Mazeiar Salehie and Ladan Tahvildari. A quality-driven approach to enable decision-making in self-adaptive software. In *Proceedings of International Conference on Software Engineering-Companion*, pages 103–104, 2007.

[109] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):14:1–14:42, 2009.

[110] Mazeiar Salehie and Ladan Tahvildari. Towards a goal-driven approach to action selection in self-adaptive software. *Software: Practice and Experience*, 42(2):211–233, 2012.

[111] Bradley Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazierr, and Jeffrey M. Barnes. Architecture-based self-protection: Composing and reasoning about denial-of-service mitigations. In *Proceedings of Symposium and Bootcamp on the Science of Security*, pages 2:1–2:12, 2014.

[112] Bradley Schmerl, Javier Cámara, Gabriel A. Moreno, David Garlan, and Andrew Mellinger. Architecture-based self-adaptation for moving target defense. Technical Report CMU-ISR-14-109, Institute for Software Research, Carnegie Mellon University, 2014.

[113] Matthew G Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of Symposium on Security and Privacy*, pages 38–49, 2001.

[114] Dan Shen, Genshe Chen, Jose B Cruz, Chiman Kwan, and Martin Kruger. An adaptive markov game model for threat intent inference. In *Aerospace Conference, 2007 IEEE*, pages 1–13. IEEE, 2007.

[115] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[116] Herbert A Simon. The new science of management decision. 1960.

[117] SIPp. Traffic Generator for the SIP Protocol. [Online.] Available: http://sipp.sourceforge.net/. [Accessed: 7- January- 2018].

[118] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998.

[119] Frank Swiderski and Window Snyder. *Threat modeling*. Microsoft Press, 2004.

[120] Ladan Tahvildari and Kostas Kontogiannis. On the role of design patterns in quality-driven re-engineering. In *Proceedings of European Conference on Software Maintenance and Reengineering*, pages 230–240, 2002.

[121] Ladan Tahvildari, Kostas Kontogiannis, and John Mylopoulos. Quality-driven software re-engineering. *Journal of Systems and Software*, 66(3):225–239, 2003.

[122] Gabriel Tamura, NorhaM. Villegas, Hausi A. Müller, João Pedro Sousa, Basil Becker, Gabor Karsai, Serge Mankovskii, Mauro PezzÃÍ, Wilhelm SchÃdfer, Ladan Tahvildari, and Kenny Wong. Towards practical runtime verification and validation of self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II, Lecture Notes in Computer Science*, pages 108–132, 2013.

[123] Security under Uncertainty: Adaptive Attackers Are More Challenging to Human Defenders than Random Attackers. Moisan, frederic and gonzalez, cleotilde. *Frontiers in Psychology*, 8(982):1–10, 2017.

[124] Axel Van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of International Conference on Software Engineering*, pages 148–157, 2004.

[125] Axel Van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[126] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311, 2003.

[127] Sheng Wen, Weijia Jia, Wei Zhou, Wanlei Zhou, and Chuan Xu. CALD: Surviving various application-layer DDoS attacks that mimic flash crowd. In *Proceedings of International Conference on Network and System Security*, pages 247–254, 2010.

[128] Qishi Wu, Sajjan Shiva, Sankardas Roy, Charles Ellis, and Vivek Datla. On modeling and simulation of game theory-based defense mechanisms against DoS and DDoS attacks. In *Proceedings of Spring Simulation Multiconference*, pages 159:1âĂŞ–159:8, 2010.

[129] Yi Xie and Shun-Zheng Yu. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking*, 17(1):15–25, 2009.

[130] Esk Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, University Virginia, 1995.

[131] Eric Yuan, Naeem Esfahani, and Sam Malek. Automated mining of software component interactions for self-adaptation. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 27–36, 2014.

[132] Eric Yuan, Naeem Esfahani, and Sam Malek. A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems*, 8(4):39:1–39:39, 2014.

[133] Eric Yuan and Sam Malek. A taxonomy and survey of self-protecting software systems. In *Proceedings of International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 109–118, 2012.

[134] Eric Yuan and Sam Malek. Mining software component interactions to detect security threats at the architectural level. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, pages 211–220, 2016.

[135] Eric Yuan, Sam Malek, Bradley Schmerl, David Garlan, and Jeffrey Gennari. Architecture-based self-protecting software systems. In *Proceedings of International Conference on the Quality of Software Architectures*, pages 33–42, 2013.

[136] Hamzeh Zawawy, Kostas Kontogiannis, John Mylopoulos, and Serge Mankovskii. Towards a requirements-driven framework for detecting malicious behavior against software systems. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pages 15–29, 2011.

[137] Saman A Zonouz, Himanshu Khurana, William H Sanders, and Timothy M Yardley. RRE: a game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):395–406, 2014.