SCHEMES FOR SURVIVING ADVANCED PERSISTENT THREATS

by

Ruchika Mehresh

August 2013

A dissertation submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfilment of the requirements for the
degree of

Doctor of Philosophy

Department of Computer Science and Engineering

The thesis of Ruchika Mehresh was reviewed by the following:

Shambhu J. Upadhyaya
Professor, Computer Science and Engineering department
Thesis Advisor, Chair of Committee

Murat Demirbas
Associate Professor, Computer Science and Engineering department
Committee Member

H. Raghav Rao
Professor, Management Science and Systems department
Committee Member

Dedication

To my husband, who believed in me and always inspired me to be my best. To the rest of my family, for their unconditional love and support.

Acknowledgments

I would like to express my deepest gratitude to everyone who made this thesis possible. First and foremost, I would like to thank my research advisor Dr. Shambhu J. Upadhyaya, who has been a continuous source of inspiration and encouragement. I owe him a great deal of debt for providing me with the opportunity to work with him and for his insightful critique and patient guidance at every step of the way. Without his guidance and persistent help, this thesis would not have been possible.

I would also like to thank my committee members, Professor Murat Demirbas and Professor H. Raghav Rao for their feedback and efforts to keep my progress on schedule. A grateful thanks to Kevin Kwiat for his generous help and guidance that helped shape this thesis to what it is today. It was a great pleasure working with him on the AFRL (Air Force Research Laboratory) projects.

Among people that contributed to this dissertation, I would like to thank Dr. Ramkumar Chinchani and Krishnan Narayanan, whose research inspired many components in my own. I would also like to thank the talented Masters students, Jairaj J. Rao and Sulaksh Natarajan, who also co-authored a publication. They took time off their busy schedules to help me with the research.

Many UB students and alumni helped me at the various stages of my graduate life. I would like to thank Dr. Mohit Virendra and Dr. Vidyaraman Sankaranarayanan who were also the teaching assistants for my first security course at UB. Their zeal for their research inspired me and they guided me a great deal. Thanks to Ameya Sanzgiri and to the rest of the research group for the very many interesting discussions and brainstorming sessions.

A special thanks to Professor James Scandale with whom I worked as a teaching assistant. He is a great person to know and to work with. I picked up many useful lessons while assisting him that I am sure will help me through life.

Last but not least, I would like to thank God Almighty for His guidance and my entire family for their love, support and encouragement- without this, I am certain I would not have been able to complete this thesis.

Table of Contents

Abstract

Mission critical systems are prevalent in the military and industry which renders them attractive as targets for security attacks. Their constantly increasing structural complexity contributes to benign faults and further facilitates malicious entities. Over the years, these malicious entities in cyber-space have grown smarter and extremely resourceful. *Advanced persistent threat (APT)* is a clear example of this growing sophistication. APTs are characterized by extreme stealth, advanced skill-set, vast resources and a markedly high success rate. In view of these circumstances, mission survivability has become an essential necessity for today's mission critical systems. Most existing survivability solutions are simple combinations of traditional security measures such as network monitoring, firewalls, etc. These solutions increase the cost of attacks but do not necessarily decrease the probability of a compromise substantially. This calls for further advancement of current mission survivability solutions.

The focus of this dissertation is the robust designing and efficient deployment of an effective mission survivability solution. Such a solution must be capable of withstanding APT, one of the hardest threats encountered in recent times. The solution presented for this purpose relies upon the understanding of *attacker intent,* *objectives and strategies (AIOS)*. AIOS aids in the designing of better recovery and adaptation procedures required for enhanced mission survivability.

This dissertation has four major parts. The first part describes the underlying attack model of the various APT attacks witnessed thus far. This generic model is then used to construct a basic centralized and hardware-supported version of the solution. The hardware-support is provided via the use of unused test-logic of the underlying processor and is used to hide and safeguard information from an adversary. This solution is based on the principles of deception and node-to-node verification.

Several attack vectors under APT are known to compromise security monitors at their target systems. This leaves the systems completely unmonitored and unprotected. The second part is a solution to ensure the tamper-resistance of such critical security components. This solution is based on the principle of coveillance and concepts derived from graph theory.

The third part combines solutions from the first two parts and redesigns a solution suitable for adoption in a distributed environment. In addition, this part explores the possible replacement of test-logic hardware with the *trusted platform module (TPM)* that is integral to the hardware of modern desktops, laptops, servers and other devices.

The fourth and final part presents an extension of our scheme that enables it to effectively and efficiently detect zero-day attacks in a production environment. This is accomplished with a carefully planned deployment and real-time customization of honeypots. In essence, this part deals with the deployment issues of deception-based solutions.

In summary, this dissertation attempts to develop a game-changing mission survivability solution. Each design choice and the techniques employed are thoroughly tested and validated via simulation and experimentation. Strong security and tamper-resistance properties, alongwith its efficiency, could make it a good survivability strategy against APT for which no good solutions currently exist.

Chapter 1

Introduction

1.1 The Problem Area

Mission critical systems are commonly employed for running essential operations in numerous domains such as military, medicine and telecommunications. Some important mission critical systems are traffic control, power, life-support systems, nuclear reactor control, etc. Lives and livelihoods depend on the correct operation of these systems. Today, critical systems have become complex, open and interconnected. These characteristics render them highly vulnerable to security attacks. Moreover, because these systems run important operations, they are often seen as high-value targets. Thus, the need for mission survivability is more pressing than ever [108].

Mission survivability "is the capability of a system to fulfill its mission in a timely manner in the presence of attacks, failures, or accidents" [32]. It does not differentiate between attacks, faults and accidents and may not even care about the root cause of a malfunction. Its sole focus is mission survival and its completion; features such as tracking the perpetrator and full-system recovery are secondary. Prioritizing goals lends a strong advantage to survivability solutions as it is easier to effectively address a limited set of requirements in a system design.

Most mission critical systems already provide a combination of fault-tolerance and security. However, *fault-tolerance* via redundancy or replication is contradictory to the notion of *security* that requires limiting the trusted computing base (TCB). Thus, normal security techniques cannot be applied to fault-tolerant systems. As a result, security practitioners employ a three-layered security solution for fault-tolerant systems. The three layers are fault/threat avoidance/prevention, detection and recovery. The first layer is fault/threat avoidance/prevention which consists of proactive measures to reduce the probability of any faults or attacks. This is usually achieved via advanced design methodologies such as encryption, firewalls and password protection. The second layer viz. fault/threat detection, primarily consists of an intrusion detection system (IDS) that attempts to detect faults or attacks that occur despite the prevention layer. The final layer is recovery that focuses on recovering the system after a fault or an attack. Generally, fault-tolerant systems rely on replication and redundancy for fault-masking and system recovery. The additional three layers of security provide a strong defense to fault-tolerant mission critical systems against malicious actors. Yet, if a determined adversary stages an attack on the recovery layer, it is quite possible that the mission will fail due to lack of any further countermeasures.

Mission survivability builds upon security and fault-tolerance features of a system. It is usually composed of four layers – prevention, detection, recovery and adaptation/evolution [32]. Adaptation is the evolution of a system state after each recovery. In other words, lessons learned from the observation of faults or attacks

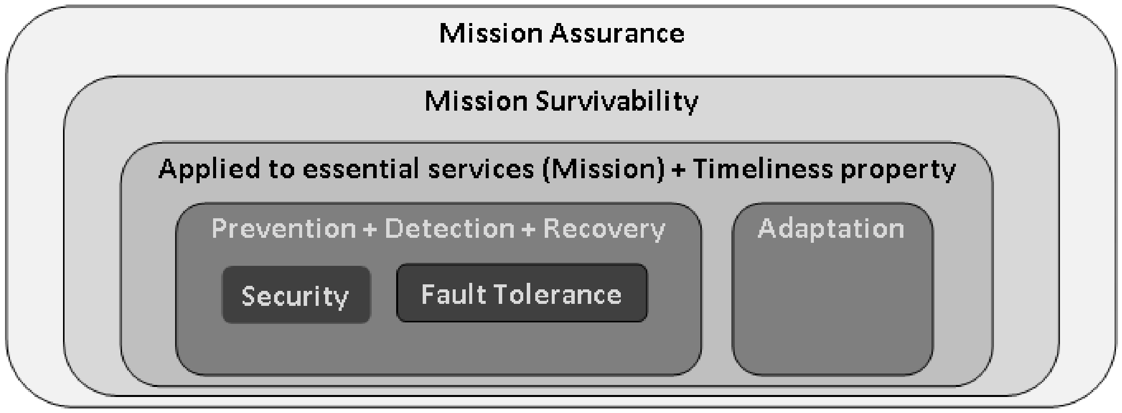
Mission Assurance
Mission Survivability
Applied to essential services (Mission) + Timeliness property
Prevention + Detection + Recovery
Security
Fault Tolerance
Adaptation

Figure 1.1: Basic architecture of a mission critical system

are applied in real-time to strengthen a system's defense.

Most existing survivability solutions simply employ an array of existing security and fault-tolerance techniques. Consequently, they inherit the drawbacks and loopholes of those techniques, along with their unreliability. Figure 1.1 explains survivability in the context of mission critical systems. Mission assurance is a generic term encompassing system engineering, risk management, quality management, etc. Mission survivability is usually considered a subset of mission assurance.

1.2 Motivation

Hacking community has been gathering momentum with dedicated forums that not only provide powerful and ready-to-use tools to script kiddies but also an excellent communication channel for professionals. Well-orchestrated cyber attacks are not uncommon in current cyber scenario. With the advent of APT, the seriousness of situation has increased manifolds. APT is "a new breed of insidious threats that use multiple attack techniques and vectors and that are conducted by stealth to avoid detection so that hackers can retain control over target systems unnoticed for long periods of time" [133]. APTs are usually characterized by extreme stealth, advanced skill-set, vast resources and a markedly high success rate. Traditional prevention, detection and recovery measures are often ineffective against them. Their strength lies in their use of deceptive measures such as trail-hiding, detection evasion, etc., along with a combination of diverse attack vectors.

Existing defense solutions, especially in a production environment, do not offer any effective countermeasures against the use of deception by intruders. Although solutions like shadow honeypots [4] provide good protection against zero-day and stealth attacks but their cumbersome anomaly detection makes them unsuitable for real-time use in production environments.

1.3 Problem Definition

Consider a generic mission critical system responsible for continuously and reliably providing a service such as space shuttle control, a sensor network for seismic event monitoring, life support system or traffic control. This service consists of n critical operations as shown in Fig. 1.2. Each critical operation and data storage module is made fault-tolerant via replication. Securing this fault-tolerant system requires the aforementioned three layers – prevention, detection and recovery. Methods and techniques of prevention (e.g., cryptographic technologies, access control mechanisms, firewalls, etc.) are chosen based on system requirements and capabilities. Detection measures usually fall under two broad categories – signature-based detectors and anomaly detectors [9]. Signature-based detectors (e.g., Tripwire [62]) work extremely well for known malware and attack patterns. Anomaly detectors (e.g., IDES [83]) are required for intrusions where the attack signatures are not known. They work relatively well for zero-day attacks and the attacks with changing patterns. However, poorly implemented anomaly detectors suffer from low

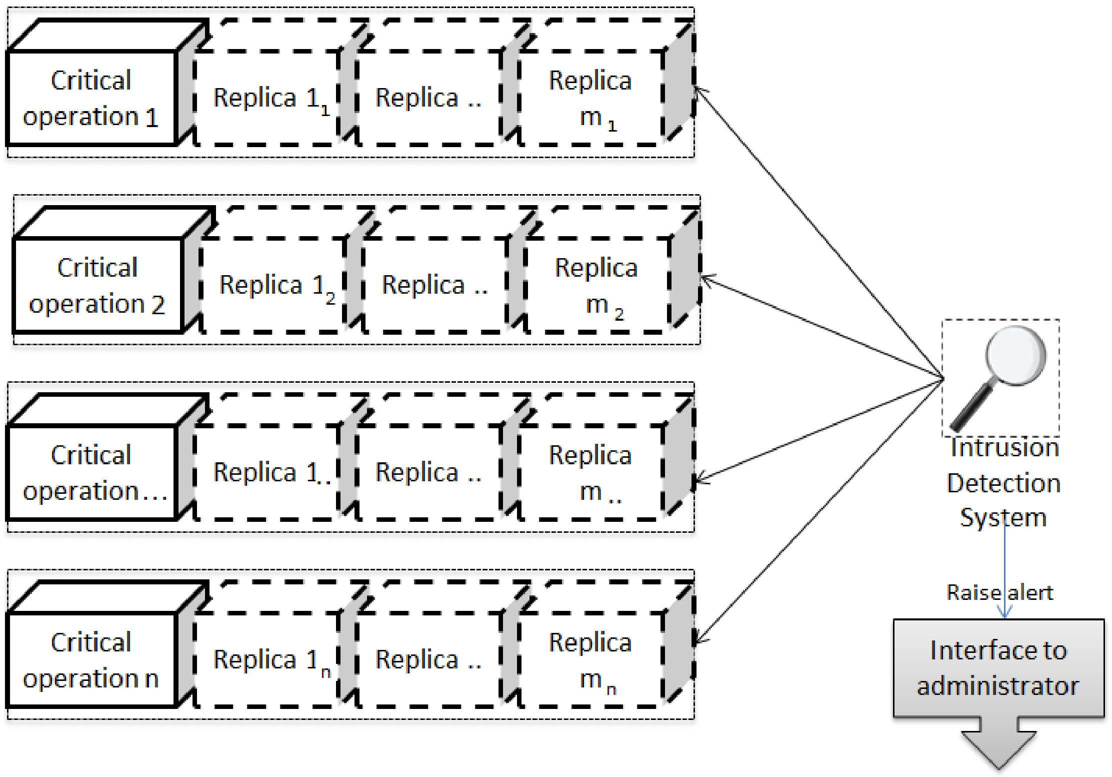

Figure 1.2: Sample mission critical system running n critical operations. Each critical operation is made fault-tolerant with m replicas. All replicas are monitored by an intrusion detection system that reports to the administrator.

accuracy and efficiency. All IDSes raise alerts on intrusion detection that are reported to the administrator via an interface like logs or system pop-ups. Recovery procedures can be either proactive or reactive in nature [127]. Their selection and implementation are system dependent. Usually, recovery is initiated based on the results from the detection layer.

Until this point, the presented mission critical system in equipped with security and fault-tolerance. These features are extended to provide mission survivability. Note that the terms *system survivability* and *mission survivability* are used interchangeably throughout this dissertation because system survivability is almost always reduced to mission survivability. System designers should choose a set of critical services that constitute the mission and ensure their survivability.

Survivability can be either static or dynamic. Under static (proactive) survivability, redundancy is used to support mission critical services. Replicas can be located in the same or different (more robust) locations. However, these static replicas can be dynamically reconfigured based on evolving survivability needs. Introducing spatial and time diversity in replicas further helps survivability. Dynamic (reactive) survivability focuses on resource management after the damage via reallocation or preemption of resources. Park and Chandramohan [104] describe the static, dynamic and hybrid survivability models for a distributed system.

As discussed previously, mission survivability adds another layer to the existing three-layered security setup of mission critical systems. This new layer, adaptation or evolution, is required so the system can survive repeated attacks of the same kind.

Survivability should not only withstand anticipated attacks but should also be able to respond to unanticipated events. Robust systems should be able to survive in all kinds of new and open environments. Thus, this dissertation especially focuses on recovery and adaptation because they help a system survive in a hostile environment against advanced and unknown attacks.

Advanced and persistent attacks use a combination of attack vectors as well as stealth. For effective APT detection, the IDS needs to monitor and correlate isolated events. This information aids in the understanding of the attacker intent, objectives and strategies (AIOS). AIOS further aids the designing (customizing) of effective recovery procedures. Recoveries are usually of two types – generic or targeted. Generic recoveries are based on approximate damage assessments. For example, if an auxiliary service is deemed compromised, a generic recovery procedure will simply switch it off. If multiple intrusions are detected, a generic recovery procedure may just restart the entire system and load the last stored checkpoint. Note that this will restore the system to the same state that was earlier compromised and is usually time-consuming. Moreover, simultaneous malfunction of multiple services may or may not be isolated events. Another drawback of a generic recovery procedure is that although it does not guarantee an intruder-free system but the intruder is made aware of the detection. Under APT, adversaries are smart enough to switch their attack strategies (or AIOS). Initiating a recovery provides the adversary with feedback that helps him better understand the system's defense and adjust his strategies accordingly. Such events strengthen the attacker and weaken the mission. On the other hand, a targeted recovery takes the approach of wait-and-watch. It gathers data about the possible footholds of an attacker within the system and his AIOS. Using this information, it is not only possible to clear intrusions all at once (completely blocking out the adversary) but predict any repercussions in case the recovery is not as effective as expected. Targeted recoveries can be extremely effective when combined with game theory in order to strengthen system survivability.

Following is a step-by-step construction of possible scenarios that threaten the survivability of the system described in Fig. 1.2:

- *Non-replicated critical services:* Designers must accurately identify all critical services and provide them with effective fault-tolerance (replication or redundancy). Replication not only provides effective prevention against faults and accidents but makes it somewhat harder for an attacker to compromise the mission.

- *Replicated services with ineffective monitoring*

  – *Byzantine fault-tolerance:* It involves state machine replication of critical services and works well for benign faults. However, in case of malicious attacks, an adversary who can compromise one replica can compromise the rest of them just as easily. If the attack is automated, then the mission survivability depends solely on the access control mechanism of each replica [24].

– *Byzantine fault-tolerance with proactive recovery:* Under proactive recovery, all replicas of a critical service are rejuvenated periodically even if there is no reason to suspect any damage. Such recoveries can usually tolerate the existence of *f* faulty replicas between scheduled recoveries. If the number of damaged replicas increases beyond *f*, effective recovery of all replicas cannot be guaranteed. In absence of diversity, it is not difficult for an adversary to compromise more than *f* replicas (within the recovery window) if he has compromised one already and has access to others. Moreover, compromised replicas can be used to disrupt system's normal operations to an attacker's advantage. For instance, compromised replicas can generate extra traffic to clog the network and delay the next round of recovery. This gains the adversary more time to compromise *f+1* replicas [127]. This is a classic case of attacking the recovery phase.

  The longer an intruder stays inside the system, the more information he gains about the system. All attempts, successes or failures, provides him with valuable insights. Thus, the probability of a successful compromise is higher for stealthy attackers in long-running missions.

– *Byzantine fault-tolerance with proactive-reactive recovery:* Proactive recoveries burden the system and add substantial overhead if scheduled too frequently. Lower frequency leaves the system vulnerable for longer periods of time. As a result, reactive recoveries were introduced. They are initiated on-demand when IDSes suspect any activity or event. However, almost all IDSes suffer from false negatives. Moreover, arbitrary faults are very difficult to detect. Hence, reactive recoveries alone cannot solve the problem [48]. Usually, reactive recoveries are used in conjunction with less frequent proactive recoveries for light-weight and effective fault-tolerance. Proactive-reactive recoveries solve several major problems except that if the compromised node is restored to the same state that was previously attacked, the attacker can compromise it again in no time [127]. Another major concern of employing reactive recoveries (even as a part of proactive-reactive recoveries) is the possibility of denial-of-service (DoS) attacks. An attacker, not capable of executing an actual attack, may still be able to invoke numerous reactive recoveries exhausting system resources.

– *Proactive-reactive recovery with diversity:* Introducing spatial diversity to replicas provides relatively better security. It can be difficult and more time-consuming for the adversary to compromise $f+1$ diverse replicas. However, it is possible to compromise all of them eventually, especially in case of stealth attacks on long-running missions. However, most existing systems are not spatially diverse and upgrading them to possess spatial diversity is usually expensive. Note that even with spatial diversity, there is still the issue that the system is restored back to the same vulnerable state.

Time diversity has been suggested to complement spatial diversity. It involves changing the system state after each recovery so that it is almost impossible to predict the current system state [18]. It is highly complex to implement time diversity as a workable solution because of the on-the-fly compatibility issues. Furthermore, updating replicas and other communication protocols after each recovery consumes considerable time and resources. A decent workable solution employing space diversity still needs a lot of work [13]. Employing efficient time diversity is a step planned too far ahead in the future.

- *Vulnerabilities in IDSes:* This is the classic problem of 'Who watches the watcher.' Several retroviruses and attack vectors under APT are known to target security monitors first, leaving the system undefended against any future attacks. Many solutions have been proposed to provide tamper-resistance to security monitors. However, many of these solutions are cumbersome, complex and even insecure. Taxonomy of these solutions is included in Chapter 4.

- *Vulnerable intrusion reporting interface:* All IDSes rely on some kind of interface to report intrusions. These interfaces can be log files, desktop pop-ups, emails, etc. and need to be secured in order for an IDS to be truly effective. For instance, an attacker can delete the log files generated by an IDS so the intrusions are detected but never reported. A good reporting interface should either be tamper-proof or be able hide detection information from unauthorized users.

- *The physical access threat:* Sometimes, system nodes are deployed in an environment where physical access to them is a highly probable threat. For instance, wireless sensor network nodes are highly susceptible to physical capture. To prevent such attacks, any changes in the physical environment of a node must be reported. A reasonable solution may involve attaching motion sensors to each node. Readings from these sensors can be used to detect an unauthorized physical access.

From the above discussion, it can be concluded that a good survivability solution must be provided with byzantine fault-tolerance and a three-layered security (viz. prevention, detection and recovery). However, existing intrusion detection techniques and recovery procedures are ineffective at large against APTs that rely on stealth. There is also a strong need to tamper-proof all critical components of a good survivability solution.

1.4 Solution Approach

Based on the discussion above, this dissertation will focus on the following areas:

- Strengthening mission survivability with focus on recovery and adaptation phases.
- Designing an effective solution to withstand APT, one of the hardest attack models. Any such solution must consider that advanced attackers are capable of switching strategies based on the observed information. Therefore, any defensive action that divulges new information should consider all possible repercussions.
- Designing the solution to counter APT's deception (stealth).
- Designing the solution to stay tamper-free throughout the mission lifetime.

- Designing the solution to work effectively for long running missions. Long running missions consume a lot of time and resources and have a lot to lose if a mission is disrupted in the later stages. Additionally, stealthy attackers have more time to hide, observe and design advanced attacks for the mission.
- Effectively evaluating the solution.
- Designing the solution to work well in all environments, centralized and distributed.
- Designing the solution to be efficiently deployable in a soft real-time production environment.

The presented solution framework is referred to as Survivable framework for cyber Warfare against Advanced Threats (SWAT) in the rest of this dissertation. Based on the focus areas specified above, the design approach to SWAT has three legs:

- *Tamper-proofing security monitors at each host:* All critical components are monitored by IDSes at all times. However, advanced attacks are capable of compromising or reducing the effectiveness of these security monitors [50, 62]. Therefore, it is essential to ensure that all security monitors and critical services stay tamper-free at all times.
- *Surreptitious and tamper-proof intrusion reporting:* A good survivability solution against APT should consider all repercussions before taking any action that reveals new system information. A half-baked reaction-plan can tip-off the attacker about the detection and trigger a change in the attack strategy. In this regard, all intrusion reporting should be kept surreptitious and tamper-free until the system figures out an appropriate reaction strategy.

- *Deception-based recovery customization:* Stealth attackers progress slowly to evade detection while devising advanced attacks. They also randomize their traces and patterns so a correlation is not apparent. Deception techniques, such as honeypots, have been used for a long time to expedite attack profile development via the generation of AIOS. Deception forces certain choices on the suspected users. These choices indicate patterns that reveal the intentions and capabilities of the attacker sooner than a regular system. AIOS information is used in the customization of targeted recovery procedures. Moreover, surreptitious reporting of intrusions provides the system with the choice to hide detection information from the attacker for as long as it wants. This data-hiding and stealth buys SWAT the time to decide on an appropriate defense strategy.

Numerous design choices in SWAT helps keep it lightweight, efficient and effective for long-running missions. The design process starts off with a centralized and simplified version of SWAT, which is later modified and extended for adoption in a distributed environment. This dissertation also presents a deployment strategy for increasing the effectiveness of SWAT against zero-day attacks.

1.5 Brief Summary of Contributions

- Identified the shortcomings in current survivability solutions that make them ineffective against APT.

- Proposed a centralized survivability framework (called SWAT) that surreptitiously detects and reports intrusions via the use of hardware. SWAT uses deception to customize effective recovery and adaptation procedures in real-time.

- Proposed a light-weight solution for tamper-resistant functioning of security monitors (and other critical components) in SWAT.
- Extended the centralized SWAT framework to work efficiently in a distributed environment.
- Proposed a multi-step evaluation procedure to quickly and effectively evaluate complex systems such as SWAT.
- Proposed a scheme for effective deployment of SWAT in production environment and to increase its effectiveness against zero-day attacks.

1.6 Dissertation Outline

Chapter 2 surveys the relevant literature for the problem area defined in Section 1.1. This survey primarily targets three crucial domains. First, it describes the major works in the field of mission survivability. Second, it presents an account of the recent major instances of cyber attacks that can be categorized under APT. This is essential in deriving a generic attack model based on APT in Chapter 3. Third, it discusses the role that deception has been traditionally playing as a tool of defense. This discussion is required in order to understand why some aspects of SWAT are rooted in deception.

Chapter 3 begins with describing a generic attack model based on APT. The challenges posed by this attack model influences the set of requirements laid down for designing SWAT. This chapter then continues to describe a centralized version of SWAT which is made fault-tolerant via replication. In this version, all replicas run in lockstep and submit their checkpoints and final results to a central authority that uses voting to identify faulty/compromised submissions. IDS running at each replica is altered to lock a signature in the test-logic hardware if suspicious activity is detected. The signature stored in the hardware is invisible to the adversary. This form of information hiding is based on the principle of deception discussed in Chapter 2. Deception is required in the design of SWAT because APTs employ stealth to hide inside the system that gains them an unfair advantage. Moreover, they use system's reactions to fine-tune their strategies in order to model more effective and targeted attacks. Therefore, any myopic defensive strategy may act as a feedback benefiting the adversary. An intelligent defense should first gather enough information about the attacks in order to understand the AIOS and then devise appropriate defense strategies in real-time. The signature hidden in the hardware is securely transmitted to the central authority which is responsible for designing such strategies. Evaluating the system described in this chapter becomes complex due to the presence of both hardware and software components. Thus, a multi-step evaluation approach is employed. It combines simulation, implementation and theoretical analysis to evaluate a complex system and produce realistic results for long-running missions. This approach to evaluate SWAT is described in detail.

Chapter 4 presents a solution to the problem of ensuring that all critical components of SWAT stay tamper-resistant throughout the mission. It proposes a cyclic watchdog solution based on the principle of coveillance. Light-weight processes are arranged in a cycle and monitor each other so that it is not possible to kill anyone without alerting some other process. Any of these processes can monitor other crucial components. Various topologies of this solution are suggested for increased security or performance. Evaluations are carried out via solution implementation in an AMD SimNow simulated multi-core environment.

Chapter 5 extends SWAT for adoption in a distributed environment. As for the centralized solution presented in Chapter 3, the solution in this chapter leverages concepts of deception (hiding) in a hardware-based security setup. In addition, it explores the possible adoption of trusted platform module (TPM) in place of test-logic circuitry. TPM provides additional processing capabilities that make it more suitable for SWAT. However, it is not present in all commercial-of-the-shelf (COTS) systems. Evaluations are conducted using multi-step evaluation approach as described in Chapter 3.

Chapter 6 discusses the tendency of APT to frequently exploit zero-day vulnerabilities. This chapter presents an architecture that employs deception at the end systems in order to defend against zero-day exploits. In addition, it addresses the problems related to deploying deception-based survivability solutions in a production environment. This is an important because most traditional uses of deception are cumbersome and complex. As before, this solution also involves on-the-fly designing of AIOS-based targeted recovery procedures. Each design choice in this architecture is supported by evidence and a detailed review of related literature.

Chapter 7 concludes the discussion by reviewing the important features of SWAT and discussing the challenges involved in developing and deploying a full-scale SWAT-based system.

Chapter 2

Related Work

2.1 Introduction

In the previous chapter, Survivable framework for cyber Warfare against Advanced Threats (SWAT) was introduced as a solution for enhanced mission survivability. This chapter surveys the literature related to the design of SWAT. Domains such as fault-tolerance, mission survivability and deception are broadly explored in order to understand the current cyber scenario pertaining to the problem of mission survivability. This chapter also discusses the recent and major events of advanced persistent attacks (APT) in order to make a strong case for their consideration.

2.2 Fault-tolerance

Fault-tolerant solutions in the existing literature employ redundancy, replication and consensus protocols. Most such solutions can tolerate up to f replica failures/compromises. Given enough time and resources, it is possible for a skilled adversary to compromise more than f replicas and subvert the entire system. Thus, a combination of reactive and proactive recovery approaches is used to keep the number of compromised replicas under f at all times [127]. As attacks get more advanced and stealthy, it gets harder to detect faulty or malicious behavior [137]. Furthermore, after compromising one replica, the adversary can easily compromise other similar replicas fairly quickly. To counter this problem, researchers have proposed spatial diversity in software. Spatial diversity can slow down an adversary but eventually compromising all diverse replicas is possible. Therefore, it was further proposed to introduce time diversity in addition to spatial diversity. Time diversity modifies the expected state of a recovered system such as its OS access passwords, open ports or authentication methods. This is to ensure that an attacker is unable to exploit the same vulnerabilities that he had exploited before [18].

Chapter 1 had discussed these measures and their shortcomings in detail.

2.3 Mission Survivability

Terms such as dependability, fault-tolerance, reliability, security and survivability are often confused with each other and have several overlapping characteristics. Their ambiguous definitions present a myriad of problems for the researchers who work in related fields. Al-kuwaiti et al. [3] presented a comparative analysis of these terms to better understand their similarities and differences. Based on this analysis, a survivable system is required to resist threats such as intentional/hostile attacks, malicious activities, random failures and accidents. The constituent attributes of survivability are reliability/availability, fault-tolerance, performance, safety and security (availability, authenticity, confidentiality, integrity). Survivable systems achieve these attributes by identifying critical services and providing them with fault-tolerance and four layers of protection, viz. prevention, detection, recovery and adaptation.

Traditional prevention, detection and recovery measures are often ineffective against APT. Thus, security researchers recommend a combination of several techniques to counter APT [29]. For instance, Tankard recommended monitoring the outgoing traffic in combination with other traditional measures (e.g., keeping the system up-to-date, using firewall, data encryption, security audits, etc.) [133]. Good security practices and awareness are always recommended [126]. However, multiple solutions when combined can become heavyweight and harm a system's performance, delay a mission or become non-scalable. They are not even guaranteed to be effective.

As discussed previously, recovery layer is considered the key to mission survival in a hostile environment. Paputungan et al. [103] presented a recovery-based model for mission survivability that focuses on system reconfiguration after an incident. The work proposed a reconfiguration algorithm that assigns redundant resources to critical services and pre-empts resources from non-critical services. Resource reallocation is based on factors such as the time and cost of reconfiguration and the number of pre-empted non-critical service resources. Many other researchers have proposed work on recovery-based survivability. For example, ERAS [139] and cluster recovery [7]. There are solutions that model systems for better survivability analysis quantitatively [93, 78, 81, 147, 124].

Park and Chandramohan [104] described the static, dynamic and hybrid survivability models for a distributed system. Carvalho [22] focused on strengthening mission survivability by intelligent allocation and replication of tasks across a network. Threat assessment is used to proactively move resources that are at higher risk. On similar lines, Carvalho et al. [23] proposed a biologically inspired approach to mission survivability for cloud computing environments.

Zuo and Panda [148] proposed a logical framework for survivability in a dynamic and real-time environment. This framework focuses on the use of strategies and tactics to thwart malicious attacks. Strategies refer to the overall plan of action designed for mission survivability while tactics focus on lower-level procedures and techniques required to implement a strategy. These concepts are illustrated via a military command and control example in this work. It also mentions four principles that should be used to decide the appropriate survivability strategies:

- *Multiple-rule principle:* Multiple survivability strategies can be used collaboratively. Each strategy should focus on a different aspect.
- *Efficiency principle:* Efficiency and effectiveness of each strategy should be derived from a cost-benefit analysis.
- *Least side-affect principle:* Survivability actions should have a minimum impact on other system operations.
- *Worst scenario principle:* Survivability strategies should be prepared for the worst-case scenarios.

Literature on mission survivability is vast and difficult to review completely. This section describes a few important solutions that directly or indirectly contribute to the design of SWAT. Several other works on mission survivability are cited (wherever relevant) throughout this dissertation.

2.4 Advanced Persistent Threats

Usually, advanced defense strategies are derivations of the lessons learned after the attacks on a system. However, in order to develop state-of-the-art security solutions, the need is to proactively observe and understand the direction in which current threats are heading. This section describes the latest trends and instances of attacks on critical systems.

Today's market forces and easy access to high-end technology have influenced the attack landscape considerably. As reported by Washington Post [98], malicious sleeper code is known to be left behind in the U.S. critical infrastructure by state-sponsored attackers. This sleeper code can be activated anytime to alter or destroy information. Similar stealth and evasion methodologies are also employed during the multi-stage delivery of malware discussed in [84] and the botnet's stealthy command and control execution model in [60]. A rising trend of stealthy and advanced malware can already be seen all around [59].

A recently published report by McAfee surveyed 200 IT executives from critical infrastructure enterprises in 14 countries [12]. The report documents cyber-security experts expressing concern about the surveillance of U.S. critical infrastructure by other nation-states. There is a stable and high number of probing reports (thousands/month) and perceived network attacks by domestic utility networks. Quoting [12], "our survey data lend support to anecdotal reporting that military in several countries have done reconnaissance and planning for cyber-attacks on other nation's power grids, mapping the underlying network infrastructure and locating vulnerabilities for future attack." The rapid adoption of technology such as smart grid (that automatically monitors and controls the electricity flow) adds to the problem of protecting these critical systems. Such systems have become highly vulnerable due to automation and remote access but majority of their operators are not as concerned about the security. The survey acknowledges the need for critical infrastructure to focus its attention on more sophisticated threats such as stealthy infiltration and cyber-extortions. This need is further heightened by the emergence of APT, the critical infrastructure's slow response to the rapidly changing threat scenario and the fact that it is a high-value target.

APT is probably the most serious threat faced by critical systems today. Daly describes APT as sophisticated cyber-attacks by hostile organizations that aim to gain access, maintain a foothold and modify data at their target systems [30]. The term 'advanced' refers to the high-quality of skill set involved in designing these attacks. The term 'persistent' is used to indicate the long presence of an attacker inside the system (for completing a mission or prolonged data collection). Advanced attackers usually adapt their strategies based on observations. These changing strategies are executed via command and control (C&C) architectures. The attacks are targeted and multi-shot. Generally, the first step is to gather intelligence such as the anti-viruses (AV) running on the system. This information helps exploit weaknesses in the defense while avoiding detection. Attackers also maintain multiple malware installations and footholds in the system. Even if a system recovers from some malware installations, there is no guarantee that the system is attacker-free. Weak forms of such guarantees may come from a complete reboot/reload of important system files. However, such measures violate the timeliness property of a mission and may restore the system to the same vulnerable state that was initially compromised.

Gh0stNet is an intelligence gathering operation (an APT) that uses a Trojan horse named Gh0st Rat [30]. Some of its features are key logging, remote terminal shell, tracking videos remotely, voice monitoring, session management, hiding traces by clearing up the systems logs, etc. It has been known to compromise at least 1,295 computers.

Another well-publicized APT, Operation Aurora, targeted 35 organizations including Yahoo, Symantec and Morgan Stanley and stole intellectual property. Aurora's multi-shot attack started with advanced social engineering and targeted emails that hosted malicious Javascript code. It mostly exploited zero-day vulnerabilities in Internet Explorer and established backdoor communication with its C&C centers via TCP port 443. It used encryption for stealth in order to avoid traditional detection systems. Since then, many similar advanced attacks have been reported on organizations like Sony, Barracuda Networks and RSA security [133]. A report by McAfee [68] described details about Aurora and the preventive measures that companies can take to protect their critical assets from similar attacks in the future.

A major example of APT is the famous malware, Stuxnet [36]. It came to light in 2010 when it targeted Iran's nuclear facilities. Its variants are one of the most serious malware threats to supervisory control and data acquisition (SCADA) systems such as power plants and gas pipelines [86]. It has the capability to reprogram programmable logic controllers (PLCs) to work to an attacker's advantage and it maintains stealth. Stuxnet sniffs for a specific configuration and remains inactive if it does not find it. Another risk-minimizing feature is its limited replication (one to three) and a specific date on which it erases itself. Once a highly advanced malware is designed to damage the military systems, its spread for industrial espionage cannot be contained. "Stuxnet is the new face of 21st-century warfare: invisible, anonymous, and devastating" [47].

Malware Flame outperformed Stuxnet in sophistication and capabilities. It installs itself on endpoints and sniffs information. This information is transmitted back to the C&C center. Based on the information received, the center decides its attack strategy. Flame has limited spread and hides its traces to ensure that it can get valuable information without raising red flags. It has the capability to monitor nearly every activity on an endpoint, including conversations utilizing Bluetooth technology. Flame's operators are highly skilled, stealthy, focused and adapt their attacks to each target [96].

Stealth is usually an underlying feature in APT. It is essential due to the long process involved in compromising a highly secured system. In essence, stealth buys attackers time to slowly progress towards their goal by reducing the risk of detection. For instance, some malware delay their activation so a pattern-recognition AV cannot associate them with their source. Another example is a compromised system contacting its C&C center by embedding information (e.g., keystrokes) in domain name system (DNS) packets [30]. Stealth in malware distribution is generally achieved via junk insertion, code recording and packing [45].

2.5 Deception as Tool of Defense

Cyber warfare is not limited to military domain anymore. Knapp and Boulton [64] reviewed the information warfare literature from 1990 to mid-2005 and made a strong case for how cyber warfare has extended to other domains such as e-commerce, telecommunications, etc. Baskerville [16] discussed the asymmetric warfare theory in context of information warfare. According to the asymmetric warfare theory, attackers have the advantage of time and stealth over the defending systems. Thus, in order to counter this imbalance, defense needs to be "agile and adaptive." Deception can be an effective tool to even out this asymmetry in the design of mission survivable systems. Defensive deception is an act of intentional misrepresentation of facts to influence an attacker's actions in favor of the system [31].

Deception itself in warfare is not new [135, 28]. However, deception has several associated legal and moral issues with its usage in today's society. The discussion of morality and legality in integral to the use of deception for defense and it is never ending. Cohen [28], the author of deception toolkit [27] discussed the moral issues associated with the use of deception throughout his work. Lakhani [69] discussed the possible legal issues involved in the use of honeypots.

The static nature of today's networks presents a sitting target and the network defense needs to grow smarter. Patch development time for most exploits is much higher than the exploit development time. Repik [116] documented a summary of internal discussions held by Air Force Cyber Command staff in 2008. His work made a strong argument in favor of using deception as a tool for degrading attackers' effectiveness. He argued why planned actions taken to mislead hackers have merit as a strategy and should be pursued further. Until recently, the focus has been mainly on the use of deception by hackers or the honeypot systems but little has been done to systematically model and examine deception-based computer security operations.

Deception theory is well documented in social and behavioral sciences. Cohen et al. [28] pointed out that there are only two ways of defending against a powerful adversary. One is to be stronger than him, and the other is to manipulate him into reduced effectiveness (by way of deception). Deception aims to influence adversary observables by concealing or not concealing as needed. Thus, as mentioned in [28], deception consists of – 1) determining what the adversary should and should not observe, 2) creating simulations to induce desired observations and control the focus of attention, and 3) using concealment to inhibit undesired observations.

Murphy [97] talked about the use of cyber deception to address the issue of ever increasing threats in today's cyber space. She discussed the techniques of deception such as fingerprint scrubbing and obfuscation. Her work is based on the principle of holding back important information from the attacker to render the attack weak. In cyber space, knowledge is everything, its impreciseness or incompleteness reduces the quality of resulting attacks. Fowler and Nesbit [38] argued for the integration of deception with regular operations as an effective warfare tactic. There is vast literature and taxonomies on the use of deception to secure computer systems and information in general [28, 117, 116].

2.6 Conclusion

This chapter surveyed the literature relevant to SWAT and highlighted the problems of mission survivable systems by describing the latest instances of APT. Because the relevant body of literature is extremely vast, only the major relevant works are included in the survey. Deception is an important part of SWAT design, therefore, the importance and use of deception as a defense technique has also been discussed.

In the next chapter, a generic APT-based attack model is described and a centralized version of SWAT is presented.

Chapter 3

SWAT - Centralized Architecture

This chapter describes a centralized version of SWAT (Survivable framework for cyber Warfare against Advanced Threats). But prior to that, an attack model is presented that will serve as the basis for its design.

3.1 Attack Model

Chapter 2 described the current attack landscape for cyber systems and presented several instances of advanced persistent threats (APT). This section builds upon it to present a generic attack model representative of APT. Mandiant corporation [85] considers the attacks under APT to have the following lifecycle:

- *Initial compromise:* Social engineering, phishing, zero-day viruses, etc.
- *Establish foothold:* Install backdoors, Trojan horses, etc.
- *Escalate privileges:* Gain administrative privileges using exploits, password cracking, etc.
- *Internal reconnaissance:* Collect confidential infrastructure information.

- *Move laterally:* Compromise more internal systems.
- *Maintain presence:* Ensure continued control over channels and credentials without raising red flags.
- *Complete mission:* Steal data or subvert the mission at an appropriate time.

This dissertation works with an attack model presented in Fig. 3.1. This attack model is an extension of the models presented by Repik [116] and Gragido [46]. The attack starts with intelligence gathering, followed by initial planning and development. Techniques such as social engineering and phishing are used by the attacker to observe and learn more about the system. Based on the information gathered, the attacker decides whether he wants to proceed with the attack. This decision depends on a variety of factors ranging from available resources to cost-benefit analysis of the attack. If the attacker decides to go ahead, he could attempt to establish multiple footholds inside the system by installing backdoors or Trojan horses and may attempt privilege escalation. Meanwhile, the attacker also performs internal reconnaissance to ensure that stealth is maintained at all times. This is because APTs involve high-stakes (state-funded or corporate espionage) and stealth becomes inevitable in these cases. Moreover, because of the massive amount of resources and time involved, each failed attack will likely be a huge loss to the attacker. Another reason why stealth is so important is because APTs need to be able to choose the time of attack to maximize its impact and minimize risk. For instance, if the attacker intends to advance a system clock by a few seconds everyday, it may be a good idea to do it at a time when there is no live monitoring. After accomplishing all sub-tasks, the attacker needs to wait until it is time to execute the final attack. Meanwhile, if the attack is detected, the attacker will most likely alter his strategy as part of the contingency plan. Contingency

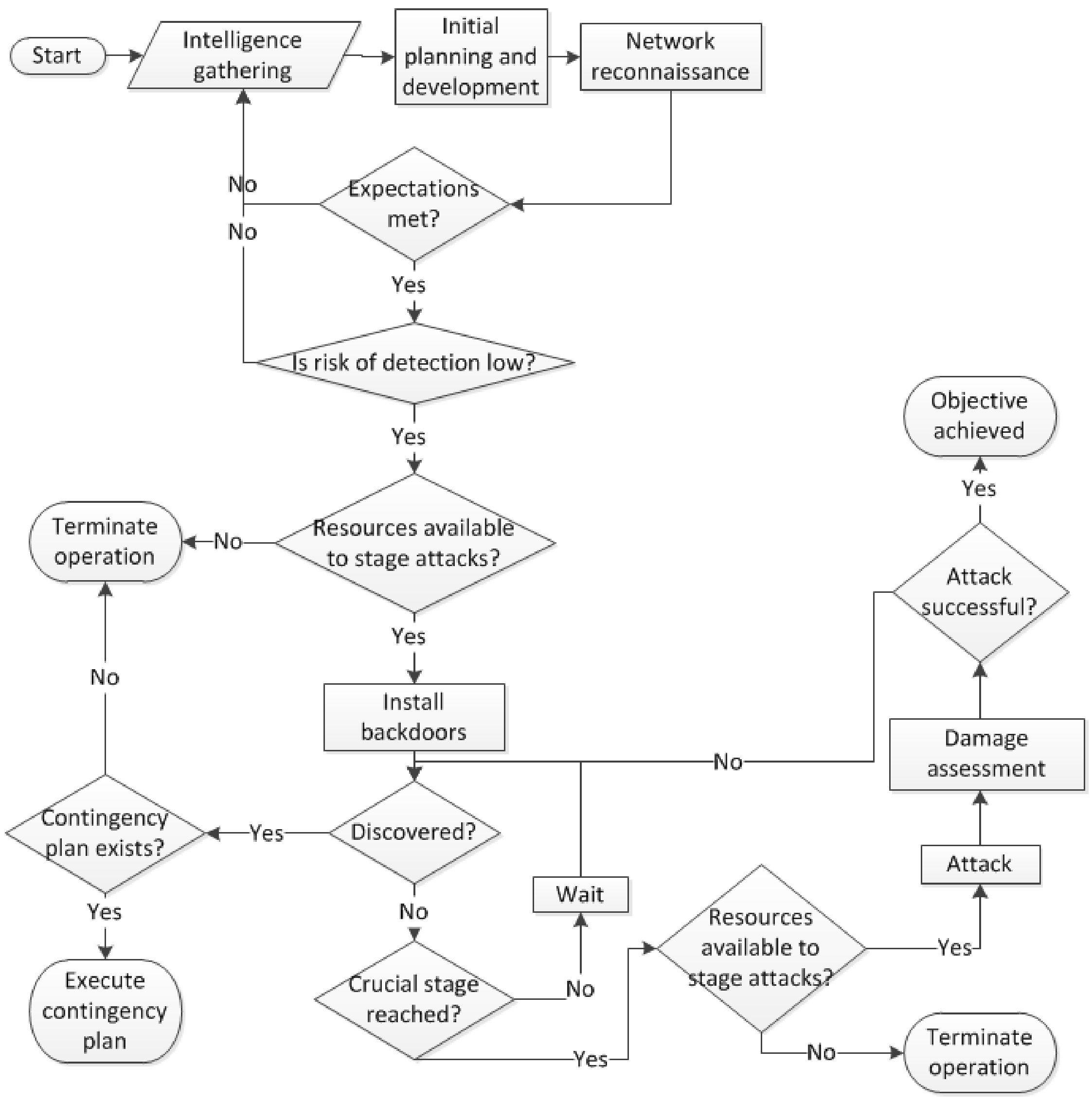

Figure 3.1: APT-based attack model

plans usually include actions such as deleting traces and system files or advancing attack timeline and crashing the system. Such events are rarely good news for the defender.

The longer an advanced attacker stays inside the system, the stronger he becomes. Sometimes the only way left is a system-wide sanitation which is very likely to disrupt the mission. Such drastic measures can cause huge financial loss to the system, especially if the mission has been running for a long time. Thus, it is necessary that stealthy attackers are made to manifest an easily detectable pattern at an early stage.

3.2 Solution Design

The attack model described in Section 3.1 is formally presented in Fig. 3.2. Let ϕ be the set of exploitable vulnerabilities for a system with state $s(t)$, where t is time. For each vulnerability ν in ϕ, the amount of resources required to exploit it is represented by $r[\nu]$. Total resources available to an attacker is $\hat{r}$. Risk associated with exploiting each vulnerability ν is $\rho[\nu]$. Maximum risk that the attacker can afford is $\hat{\rho}$.

3.2.1 Requirements

Based on the attack model presented above, the following is the set of requirements that a good survivability solution should satisfy:

- Must fulfill all requirements of mission survivability. It should be lightweight to conserve mission's timeliness property. However, predictable delays are allowed if accounted for in the original mission timeline.

- Should be platform-independent and support the commercial-off-the-shelf (COTS) paradigm.

- Should resist APTs effectively. Defensive actions should be taken after due consideration of their repercussions. Consequently, all intrusions must be reported in a surreptitious and tamper-free manner.

```
while TRUE do
  while $\phi$ = NULL AND $\forall\nu$,$\rho[\nu] \geq \hat{\rho}$ do
    Gather intelligence
    Develop exploits
    Perform network reconnaissance
    Update vulnerability set $\phi$
  end while
  if $\exists\nu$, (r[$\nu$]$\leq \hat{r}$ AND $\rho[\nu] \leq \hat{\rho}$) then
    Install backdoors; Update $\hat{r}$
    while s(t) $\neq$ ATTACK_DISCOVERED do
      if s(t) $\neq$ CRUCIAL_STAGE then
        WAIT
      else if $\exists\nu$, (r[$\nu$]$\leq \hat{r}$ AND $\rho[\nu] \leq \hat{\rho}$) then
        Attack and exploit $\nu$; Update $\hat{r}$; Assess damage
        if s(t)=COMPROMISED then
          Operation successful and Exit
        end if
      else
        Terminate operation
      end if
    end while
    if Contingency plan exists then
      Execute contingency plan
    else
      Terminate operation
    end if
  else
    Terminate operation
  end if
end while
```

Figure 3.2: Pseudocode - Operational details of APT-based attacks

- Should be able to counter the advantage gained by advanced attackers that employ stealth. This can be achieved by forcing or manipulating the intruders into leaving discernible traces.

The SWAT version presented here is novel, centralized, recovery-focused and mission survivable. It explores the potential of utilizing a processor's test-logic for secure and surreptitious storage of intrusion detection information. This choice lends cost efficiency and deception capabilities to the design. An attacker has no way of knowing that his presence has been detected until the system responds to the detection or the attacker has access to the intrusion reporting channel. Because intrusion reporting is made surreptitious in SWAT, the system can deceive the attacker into believing that he has not been detected. This false belief can be exploited by the system to strengthen its defense. For better understanding of this architecture, a proof-of-concept mission critical system is studied and evaluated. Finally, the security, usability and performance overheads of SWAT are analyzed using a multi-step evaluation approach.

3.2.2 Assumptions

For the sake of simplicity, proof-of-concept mission critical system assumes no spatial or time diversity. SWAT is diversity-independent and should work well with systems that are either spatially or time diverse.

It is assumed that the network can lose, duplicate or reorder messages but is immune to partitioning. The coordinator (central authority, also the trusted computing base (TCB)) is responsible for periodically checkpointing all the critical components in order to maintain a consistent global state. A stable storage at the coordinator holds recovery data through all the tolerated failures and their corresponding recoveries. Sequential and equidistant checkpointing is assumed [33].

Replicas are assumed to be running on identical hardware platforms. Each node has advanced central processing unit (CPU) and memory subsystems along with the test-logic hardware (in form of design for testability (DFT) and built-in

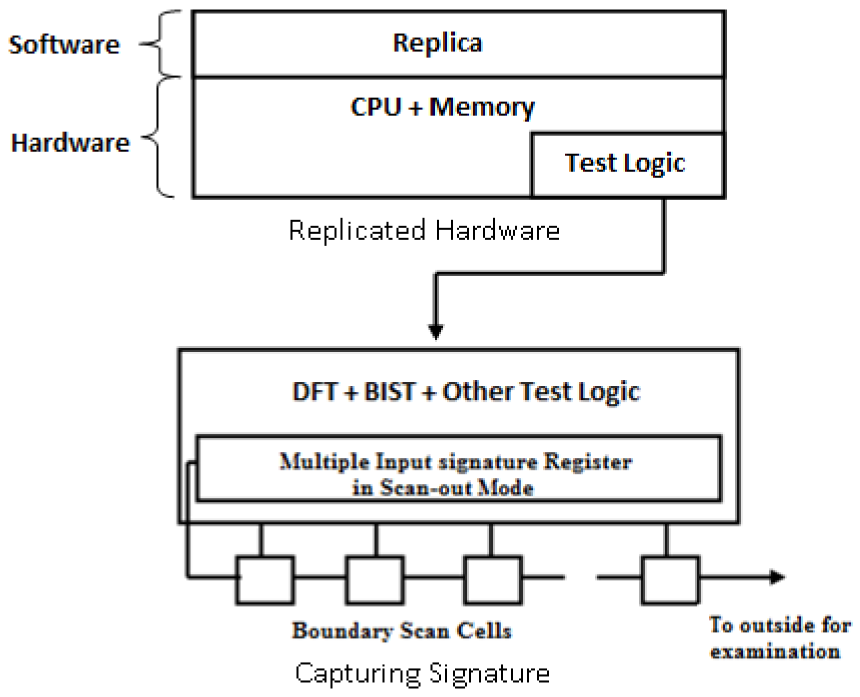
Software
Hardware
Replica
CPU + Memory
Test Logic
Replicated Hardware
DFT + BIST + Other Test Logic
Multiple Input signature Register in Scan-out Mode
Boundary Scan Cells
To outside for examination
Capturing Signature

Figure 3.3: Hardware used by SWAT for secure and surreptitious storage

self-test (BIST)) that is generally used for manufacture test. All chips comply with the IEEE 1149.1 JTAG (Joint Test Action Group) standard [2]. Figure 3.3 elaborates the test-logic and boundary scan cells corresponding to the assumed hardware.

This chapter focuses on using test-logic for secure storage. Though, trusted platform module (TPM) [134] can achieve all the same objectives but many COTS processors are not TPM-equipped.

A software tripwire is running at each replica and detects a variety of anomalies. By instrumenting the openly available tripwire source code [52], an intrusion alert/alarm (called *integrity-signature* in the sequel) can be directed to a set of system registers using low-level coding. The triggered and latched integrity-signature is read out by taking a snapshot of system registers using the scan-out mode of the observation logic associated with the DFT hardware. The bit pattern is brought out to the CPU ports using the IEEE 1149.1 JTAG instruction set in a tamper-resistant manner. Once it is brought out of the chip, it can be securely sent to the coordinator for verification and further action. This way, the system is able to surreptitiously diagnose an adversary's actions.

3.2.3 Basic Design

SWAT relies on deception-based recovery to ensure that the attacker is not alerted of detection until the system is ready to respond effectively. Concurrently, the system gathers more knowledge about the attacker's presence inside the system and the AIOS (attacker intent, objectives and strategies). At some later point, when the attacker launches the final attack, it fails because the corrupted replicas (although kept operational) have been identified and isolated by the system. Thus, the mission no longer depends on these replicas. SWAT requires at least two correctly working replicas to provide a duplex system for the mission to succeed. In the worst case, when all replicas are compromised, the system will not deliver a result instead of delivering a wrong one. This is a necessary condition for many safety-critical applications. Alternatively, if the system has gathered extensive knowledge about the intruder's footholds inside the system, it can design and initiate a recovery procedure. Such a customized recovery is more effective in blocking the attacker out of the system by recovering all the footholds at once. Recoveries based on incomplete information may only recover a few footholds and alert the attacker. This can lead the attacker to switch strategies and weaken the system's defense.

In an attempt to hide detection information from the adversary, it is required that the intrusion reporting channel be made inaccessible to him. However, if an adversary compromises a replica by gaining root privilege to user-space components, one should note that any solution developed in user-space cannot be expected to remain hidden and tamper-resistant. Therefore, SWAT achieves detection of node compromise through a verification scheme implementable in low-level hardware. Software or hardware-driven tripwires are used to help detect any ongoing suspicious activity and trigger a hardware signature that indicates the integrity status of a replica. This integrity-signature is generated without affecting the application layer and hence the attacker remains oblivious of this activity. Additionally, an advanced attacker is not likely to monitor the system thoroughly as that may lead to his own detection.

3.2.4 Checkpointing

In the proof-of-concept prototype for SWAT, the checkpointing module that affiliates to the coordinator establishes a consistent global checkpoint. It also carries out voting procedures that lead to anomaly detection due to faults, attacks or both.

The coordinator starts checkpointing/voting process by broadcasting a request message to all the replicas and asking them to record checkpoints. It also initiates a local timer that runs out if it does not receive expected number of replies within a specific time frame. On receiving the *request message*, all replicas pause their respective executions and take a checkpoint. These checkpoints are then submitted to the coordinator (over the network) through a secured channel using encryption. On receiving the expected number of checkpoints, the coordinator compares them for consistency. If all checkpoints are consistent, it broadcasts a *commit message* that completes the two-phase checkpoint protocol. After receiving the commit message, all replicas resume their respective executions. This is how all the replicas run in lockstep. In case that the timer runs out before the expected number of checkpoints is received at the coordinator, it sends out another request message. All replicas send their last locally stored checkpoints as a reply to this repeated request. In this application, the allowed number of repeated requests caused by a non-replying replica is limited to a maximum of three. If a replica fails to submit the checkpoint three times consecutively, it is assumed dead. A commit message is then sent by the coordinator to the rest of the replicas if their checkpoints are consistent. In case that checkpoints are not consistent, the coordinator replies with a *rollback message* to all the replicas. This rollback message includes the last consistent checkpoint that was stored in the coordinator's stable storage. All replicas return to the previous state of execution as specified by the rollback message. If a certain replica fails to deliver consistent checkpoint and causes more than three consecutive rollbacks, the fault is considered permanent and the replica is excluded from the system.

The integrity signature generated at each replica piggybacks the periodic checkpoints that are being sent to the coordinator. This signature quantifies the integrity status of a replica since its last successful checkpoint. For simplicity, the values used are – all-0s (for an uncompromised replica) and all-1s (for a compromised replica). If the coordinator finds any integrity-signature to be all-1s, then the corresponding replica is blacklisted and any future results/checkpoints from it are ignored at the coordinator. However, the coordinator continues normal communication with the blacklisted replicas in order to keep the attacker unaware of the detection.

Finally, all results from each of the non-blacklisted replicas are voted upon by the coordinator for generating the final result.

3.3 Performance Analysis

It was concluded earlier that APTs are the most potent in case of long-running missions. Therefore, for the performance analysis of SWAT, the focus is especially on missions that run for a long time.

SWAT employs built-in hardware for security and the integrity-signatures piggyback the checkpoints. Thus, the security comes nearly free for systems that already employ checkpointing for fault-tolerance. However, many legacy systems that do not use any checkpointing will need to employ checkpointing before they can benefit from this scheme. In such cases, cost of checkpointing is included in the cost of employing SWAT. The following three cases represent all the possible scenarios of adopting SWAT for existing systems:

- *Case 1:* Includes all the mission critical legacy systems that do not employ any checkpointing or security protocols.
- *Case 2:* Includes all the mission critical systems that already employ checkpointing. This case assumes the absence of any failures or attacks. Note that it is the worst case scenario when a system jumps from being a Case 1 to being a Case 2. Such a system will employ checkpointing in the absence of any faults/attacks. Hence, they end up paying the price without any benefits. However, if a system is already a Case 2, SWAT adoption comes for free.
- *Case 3:* Includes all the mission critical systems that employ checkpointing and SWAT, both. This case assumes the presence of failures and attacks.

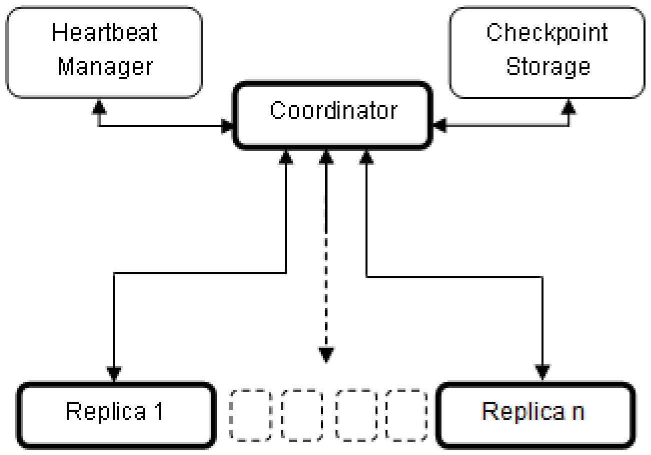

Figure 3.4: Overall system design

Figure 3.4 shows a mission critical application with n replicas. Replicas are identical copies of the workload executing in parallel in lockstep. This application is assumed to be running SWAT. Thus, the coordinator is the core of this centralized and replicated system and is responsible for performing voting operations on intermediate results, integrity-signatures and checkpoints obtained from the replicas. The heartbeat manager broadcasts periodic ping messages to determine if the nodes are alive.

Since SWAT is composed of hardware and software subsystems, one standard simulation engine cannot suffice. Therefore, results obtained from individually simulating the software and hardware components are combined using the multi-step evaluation approach.

3.3.1 Multi-step Evaluation

Multi-step approach is required for the evaluation of SWAT (as mentioned in the previous section) because there are no existing benchmarks for such complex systems. Multi-step evaluation provides a combination of theoretical analysis, pilot system implementation and simulation in order to deliver more realistic and statistically accurate results. This mix of evaluation techniques can be optimized to obtain an evaluation procedure that minimizes the development effort and cost (resources and time) while maximizing the accuracy and efficiency of the evaluation.

As new techniques of fault-tolerance and security emerge, so does the need for suitable tools to evaluate them. Usually, system security is verified via logical test cases but the performance of security algorithms needs to be quantitatively measured. The diversity of systems makes it difficult for researchers to decide on a standard, affordable and openly available simulation tool, evaluation framework or experimental test-bed. All evaluation approaches have certain merits and several drawbacks. For instance, evaluation using a system prototype has higher accuracy but it is not as scalable as the simulation. Many times, it is inefficient to evaluate a new system using any of the available tools. However, to modify the existing tools involves a lot of effort. Researchers may also choose to develop their own tools (theoretical models, simulations, etc.) but a wrong approach can adversely affect the cost of development and accuracy of results. For instance, the validity of simulation results cannot be established unless the system model is thoroughly validated and verified.

This section presents the multi-step evaluation approach for mission survivable systems. The system is modeled based on a divide-and-conquer approach that allows the use of different evaluation tools at different levels of granularity. The multi-step approach tries to strike a balance between efficiency, effort, cost and accuracy of an evaluation procedure.

Researchers usually choose from one of the three methods to evaluate a system based on its current state of development. The first method is continuous time markov chains (CTMC), which is used when the system architecture has not been established yet [43]. When system architecture is available, the second method viz. simulation can be used to model its functional behavior. The third method involves conducting experimentation on a real-world system prototype. Relying solely on each of these methods has both advantages as well as disadvantages. Based on the system type and its current state, choosing a wrong tool may increase the complexity and cost of evaluation and decrease the accuracy. For instance, if a system prototype is developed for evaluation, results will be more accurate and representative of the real-world conditions such as hardware faults and network conditions. However, system prototypes are usually not very scalable. They may be expensive or even impossible to develop. In such cases, simulation is preferred because it is easier to develop and scales well. Simulation enables the study of feasibility, behavior and performance without the need of an actual system. It can also run at any speed compared to the real-world and thus can be used to test a wide range of scenarios in a short period of time. However, accuracy is a major issue in simulated system models. Designing a highly accurate and valid simulation model is not only difficult but sometimes costly in terms of time and resources.

Multi-step approach is a combination of three concepts – *Modular decomposition, modular composability and parameterization* [90]. Modular decomposition consists of breaking down a problem into smaller elements. Modular composition involves production of elements that can be freely combined with each other to provide new functionality. Parameterization is the process of defining the necessary parameters for evaluation. A modular functional model of the system can be developed by using these three concepts. Such a model is hierarchical in terms of the level of detail/granularity. It starts at the highest level of abstraction and works downwards toward a more detailed level/finer granularity. Each level is recursively decomposed whenever possible. When the maximum level of decomposition is reached, this level is then replaced with a black-box. This black-boxed level is too complex to model (or further decompose) but it can be easily described using stochastic variables (fault rate, bandwidth, etc.) derived statistically from experimentation. This level can also use theoretical analysis such as queuing theory.

The basis of multi-step approach is the observation that a simulation model develops down the levels of granularity of detail (becomes finer) and the development of system prototype/theoretical analysis goes up the levels of granularity of detail. They meet in the middle where the accuracy, simplicity and efficiency of this approach can be maximized. Refer to Fig 3.5. Since the higher levels of this model are simpler to design, they are less prone to design errors. Hence, simulation modeling starts from the highest level of abstraction and develops downwards to finer levels of granularity, adding more complexity and detail to the design. System prototype or theoretical analysis is easier to handle at the lowermost levels of detail. For instance, a prototype can easily deliver data about the network traffic but modeling the factors that affect traffic flow is very difficult. Hence, the prototype development moves upwards towards a coarser level of granularity. Adding more functionality to a prototype (in moving upwards) increases the cost. Thus, a designer can concurrently make these two progressions to meet in the middle where the accuracy, simplicity and efficiency of the evaluation can be maximized. Refer to Fig. 3.5.

Multi-step evaluation of SWAT has three components – Java implementation based on Chameleon [58], Arena simulation [8] and Cadence simulation. Arena is a discrete event simulator and is used at the highest level of abstraction. This

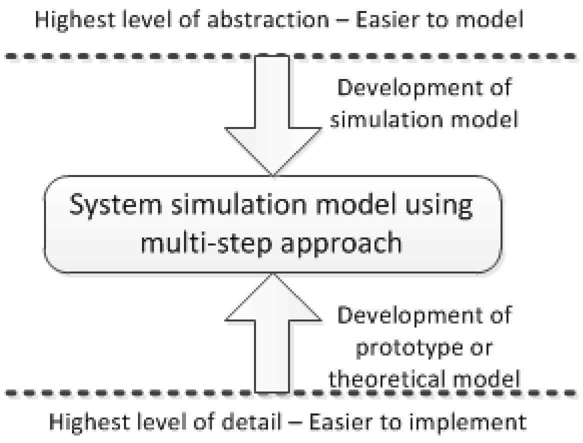
Highest level of abstraction – Easier to model
Development of simulation model
System simulation model using multi-step approach
Development of prototype or theoretical model
Highest level of detail – Easier to implement

Figure 3.5: Development process of simulation model using multi-step approach

choice is also beneficial in working with long-running missions because conducting real-time experiments for long-running missions is not efficient. The lower levels of abstraction that become too complex to model are black-boxed and parameterized using a Java-based system prototype. Note that this Java prototype does not implement all system functionalities. It only implements the functionalities that generate data required to parameterize the black boxes. The Java prototype uses socket programming across a network of 100 Mbps bandwidth. Experiments are conducted on a Windows platform with an Intel Core Duo 2 GHz processor and 2 GB RAM. Cadence simulation is primarily used for the feasibility study of the proposed hardware scheme. To verify the precision of the employed simulators, test cases were developed and deployed for the known cases of operation.

The Java prototype accepts workloads and executes them in a fault-tolerant environment as shown in Fig. 3.4. Java SciMark 2.0 workloads used in these experiments are – fast fourier transform (FFT), jacobi successive over-relaxation (SOR), sparse matrix multiplication (Sparse) and dense LU matrix factorization (LU). These are the standard large data sets [91]. Data is collected from the prototype runs and fitted probability distributions are generated using Arena's input data analyzer. These distributions define the stochastic parameters for the black-boxes of the model. Once the model is complete, it is simulated in Arena.

Feasibility of the hardware component of SWAT is examined as follows – The integrity-signature at a replica is stored in the flip flops of the boundary scan chain around a processor. This part of the simulation is centered on a boundary scan inserted DLX processor [106]. Verilog code for the boundary scan inserted DLX processor is elaborated in Cadence RTL compiler. To load the integrity-signature into these scan cells, a multiplexer is inserted before each cell. This multiplexer has one of the inputs as test data input (TDI) and the other from the 32-bit signature vector. Depending on the select line, either the test data or the integrity-signature is latched into the flip flops of the scan cells. To read the integrity-signature out, bits are serially shifted from the flip flops onto the output bus.

3.3.1.1 Simulation Preliminaries

Most of the feasibility, behavior and performance studies for fault-tolerant systems use theoretical analysis, actual system implementations or simulations. One of the earliest attempts to develop fault-tolerant systems called software implemented fault-tolerance (SIFT) was completely software-based and used loose synchronization of processors and memory [143]. Since then, many tools and frameworks such as Chameleon [58], Globus [37], etc. have been proposed to develop and evaluate fault-tolerant systems. Apache Hadoop [70] is a Java software framework that can be used to develop data-intensive, fault-tolerant and distributed applications.

Many simulation tools and languages also exist for developing and analyzing fault-tolerant system models. CSIM [122] is a process-oriented and discrete-event simulation language that enables quick construction of concise system models. OMNeT++ [109] is a C++ simulation library and framework. Mobius [1] is a software tool used for modeling the behavior of complex systems. GridSim [115] is a grid simulation toolkit for resource modeling and application scheduling for parallel and distributed computing.

DEPEND [44] provides an integrated design and fault-injection environment for system level dependability analysis. Fault-injection can be defined as a stochastic process in DEPEND so it emulates a real world scenario.

3.3.1.2 Rationale for Using Multi-step Evaluation

Simulation can only approximate the behavior of a real system. In a real system, the components (memory, processor speed, network bandwidth, etc.) have complex inputs, interconnections and dependencies that are not always easy to model. Furthermore, two similar components such as two processors can have different behaviors even if they are modeled the same for the purpose of simulation. These factors introduce disparity between the results obtained from simulation and from experimentation. There exist several general-purpose simulators that allow the designing of stochastic system models. Stochastic parameters/variables can take into account the unpredictability of real-world. However, this approach presents the challenge of specifying the stochastic system parameters and variables such as probability distributions and seeds. Most such values for defining system parameters and variables are taken from prior projects, sometimes without proper justification or verification or are simply assumed.

The differences between simulation and experimentation results can be ignored if only approximate comparisons are required; for instance, observing a linear or exponential relationship between the input and output quantities. However, if the objective is to obtain the results as close to real-world as possible, then the simulation model needs to be realistically parameterized.

Most researchers validate their simulation models by comparing the simulation results with the real-world. However, the actual system may or may not exist. Thus, there is a need to simplify the simulation model so it can be easily verified for logical validity. Adding excessive details to a model makes it complex to understand and highly prone to design errors. For instance, in designing a network application, an attempt to design the various time-variant factors that affect network performance is extremely difficult and will most likely lead to design errors. Therefore, simulation designers usually make simplifying assumptions such as the availability of a 100Mbps network bandwidth at all times. However, the application rarely gets to use the entire bandwidth. So the time performance obtained by using such a network simulation is always optimistic than in real-world. Designers generally go to a specific level of granularity in simulation design and then make assumptions beyond that level. Multi-step approach tries to realistically estimate these assumed values. This provides more statistically accurate results along with a simpler simulation model that is less prone to design errors.

Another reason for proposing multi-step approach is to deal with the long or unbounded execution times. In many cases, there is a system prototype available where the runtime is directly proportional to some parameter/variable such as the workload size. Such experiments can run for days and are very inefficient.

3.3.1.3 Modeling System Using the Multi-step Approach

In multi-step evaluation of SWAT, the prototype is developed in Java. The simulation is discrete-event which is generally of three types – event-oriented, activity-oriented and process-oriented. Since the functional model of SWAT (to be used for simulation) contains modules that undergo processing and pass on the processing control to other modules, process-oriented simulation seemed to be the most intuitive choice. Though, event and activity oriented simulations could also be used for this purpose but the authors chose process-oriented simulation due to the simplicity of its adoption for SWAT. Workload is defined as an entity with attributes such as size, arrival time and checkpoint rate. The various stages of processing such as network, replica execution and heartbeat management are modeled as separate processes. There are a variety of tools (for instance, CSIM, JavaSim and ARENA) that can be used to execute this simulation model. However, Arena is chosen because of its user-friendly drag-and-drop interface [8].

At an abstract level the system is simple but as the granularity increases, complexities arise. The system model is created piece-by-piece and where it gets too complicated or unpredictable, that level of detail is black-boxed. This black-box is then parameterized using the experimentation with Java prototype. Note that there is no need to develop the entire system for these parameters/values.

Figure 3.6 shows the simulation model. Three main modules exist at the highest level of abstraction – network, coordinator and replica. Each module is inspected to see if it can be further decomposed into sub-modules. To verify if a new level of hierarchy can be defined, the potential sub-modules are investigated for the following two properties:

- *Composability:* The functionalities of the candidate set of sub-modules can be combined to provide the functionalities of its parent module.
- *Sufficiency:* The functionalities of the candidate set of sub-modules collectively describe the entire set of functionalities of its parent module.

The first module 'coordinator' can have three sub-modules by design – *heartbeat manager, checkpoint manager and majority voter.* These three sub-modules collectively describe the entire set of functionalities provided by the coordinator.

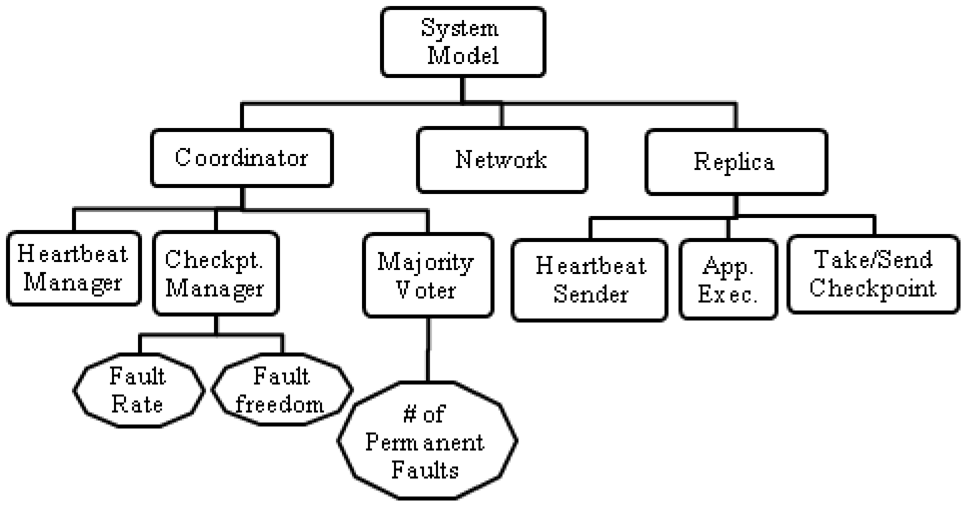

Figure 3.6: System model using the multi-step approach

Therefore, they are composedly sufficient to describe the coordinator. Hence, one additional level of the hierarchy is added for the coordinator module. The second module represents the network that is unpredictable and is complicated to decompose further. In the end, the system is modeled as a tree of height 4 is with the lowest level that cannot be further decomposed.

3.3.1.4 Parameterization

For parameterizing this simulation, the Java prototype is run several times. Data recorded from these runs is converted into probability distributions via data analysis tools (in this case, Arena Input Analyzer). It generates the best possible probability distribution for the data. Tests such as Chi-square are conducted to find out how well the selected distribution fits the data. These distributions then parameterize the black-boxed components in the simulation model.

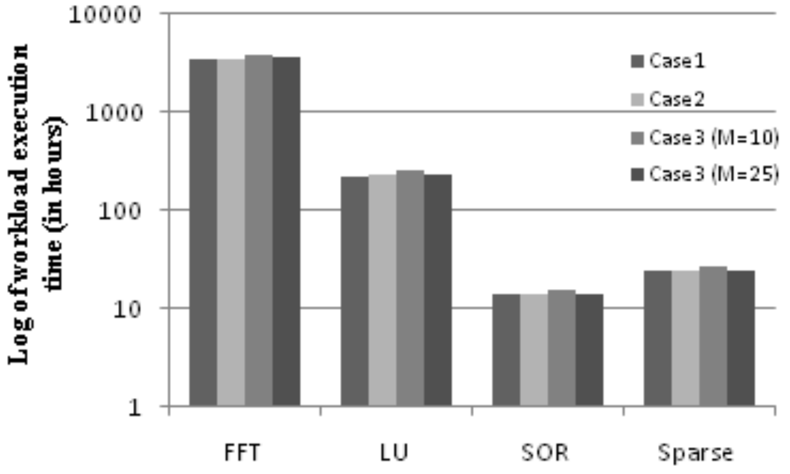

Figure 3.7: Execution times for Scimark workloads across three cases on a logarithmic scale

3.3.2 Results

To evaluate the performance of SWAT in the worst-case scenario, checkpointing overhead should be maximum. Hence, sequential checkpointing is chosen [33].

For the following analysis (unless otherwise mentioned), checkpoint interval is assumed to be 1 hour. Table 3.1 presents the execution times for the four Scimark workloads. The values from Table 3.1 are plotted in Fig. 3.7 on a logarithmic scale. It can be observed that the execution time overhead increases a little when the system shifts from Case 1 to Case 2 (i.e., employing the proposed scheme as a preventive measure as described in Section 3.3). However, the execution time overhead increases rapidly when the system moves from Case 2 and Case 3. The execution overhead will only increase substantially if there are too many faults/attacks present, in which case it is worth the fault-tolerance and security that comes along. As can be seen from the values in Table 3.1, an application that runs for 13.6562 hours for Case 1 incurs an execution time overhead of only 13.49 minutes in moving to Case 2.

Table 3.1: Execution Times (in hours) for the Scimark workloads across three cases

	FFT	LU	SOR	Sparse
Case 1	3421.09	222.69	13.6562	23.9479
Case 2	3477.46	226.36	13.8811	24.3426
Case 3 M=10)	3824.63	249.08	15.2026	26.7313
Case 3 (M=25)	3593.39	233.83	13.8811	24.3426

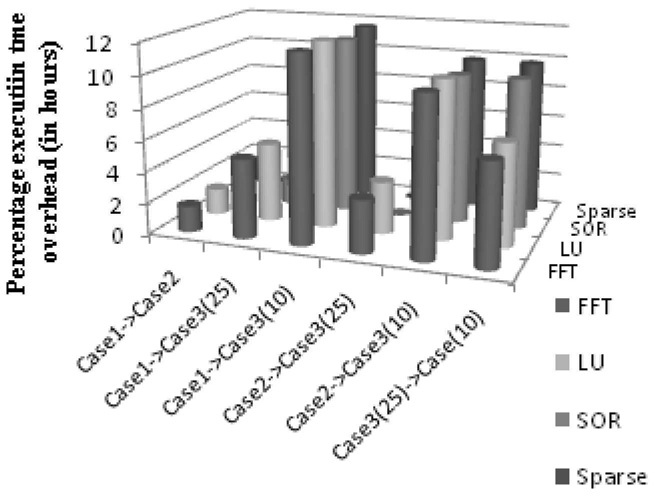

Figure 3.8: Percentage execution time overheads incurred by the Scimark workloads when shifting between cases

Figure 3.8 shows the percentage increase in execution times of various workloads when the system upgrades from a lower to a higher case. It is assumed that these executions do not have any interactions with external environments. The percentage increase in execution time is only around 1.6% for all the workloads when system upgrades from Case 1 to Case 2. The overhead for an upgrade from Case 1 to Case 3 (with mean time to fault, M = 10) is around 9%. These percentages indicate acceptable overheads.

As shown in Table 3.1, for a checkpoint interval of 1 hour and M = 10, the workload LU executes for approximately 10 days. Figure 3.9 shows the effect of increasing checkpoint interval for workload LU for different values of M ranging

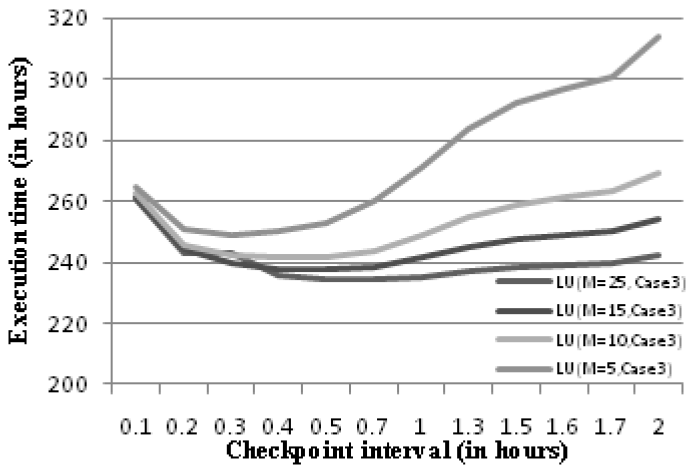

Figure 3.9: Effect of checkpoint interval on workload execution times at different values of M

Table 3.2: Approximate optimal checkpoint interval values and their corresponding workload execution times for LU (Case 3) at different values of M

	M=5	M=10	M=15	M=25
Optimal Checkpoint Interval (hours)	0.3	0.5	0.65	0.95
Execution Times (hours)	248.97	241.57	238.16	235.06

from 5 to 25. The optimal checkpoint intervals (and the corresponding execution times) for the graph plotted in Fig. 3.9 are provided in Table 3.2.

Note that these results do not just represent the data trends but are also close to the statistically expected real-world values.

3.4 Conclusion

This chapter has described an attack model rooted in APT. It has also presented a centralized version of SWAT capable of surviving such an attack model. Centralized SWAT uses test-logic hardware for tamper-proof and surreptitious intrusion reporting. Combined with principles of deception, this helps in strengthening recovery for enhanced survivability. SWAT is lightweight and does not have any application-specific dependencies. Thus, its implementation has the potential to be application transparent.

Performance evaluation employs multi-step approach. The cost analysis described various deployment scenarios for SWAT (including its application to legacy systems with no existing fault-tolerance). The evaluation shows promising results. Execution time overheads are observed to be small when faults are absent. As the fault-rate increases, the overhead increases as well. However, this additional overhead comes with strong fault-tolerance and security. Overall, SWAT is believed to provide strong survivability at low cost for mission critical applications.
Two aspects that need further exploration in this design are –

- *How to ensure that security monitors trusted to report integrity-signatures stay tamper-resistant at all times?* A solution for this is presented in Chapter 4.

- Test-logic circuitry can only be used for storage and not processing. Node-to-node verification requires the communication to be encrypted. *How to encrypt this communication and how to ensure that the component that handles encryption stays tamper-resistant at all times?* Use of TPM [134] is explored in Chapter 5 for this purpose.

In the next chapter, we explore ways to provide tamper-resistance to security monitors and other crucial services. In effect, we work towards solving the problem of ‘Who watches the watcher.’

Chapter 4

Tamper-resistant Security Monitoring

4.1 Introduction

Chapter 3 presented an advanced persistent threat (APT)-based attack model and a centralized version of SWAT (Survivable framework for cyber Warfare against Advanced Threats) to withstand it. In the concluding discussion, the issue of providing tamper-resistance to security monitors was discussed. In this regard, a lightweight solution is presented in this chapter that can provide tamper-resistance to the critical components (such as the security monitors) employed by SWAT.

Most widely deployed security solutions are written as software. This trend is encouraged by factors like convenience of use, expenses associated with updating hardware, etc. Software-based security solutions often have complex design and a large code base that renders them difficult to be tested thoroughly and completely. Consequently, they have a high probability of containing exploitable software bugs. In addition, software now-a-days is expected to satisfy a plethora of requirements, encouraging trends such as outsourcing, off-shoring and reusing open-source software. This leads to serious security loopholes in the resulting software [79, 61]. Many studies document this growing software unreliability and an increasing malware sophistication that can exploit them [15, 123, 66, 19, 73]. In general, the software industry is dominated by a few vendors (for instance, Microsoft, Apple, Oracle, Linux, Mozilla, Red Hat and Google). Even the security solutions market is mostly owned by a few leaders such as Avast, Avira, Symantec, AVG and McAfee [102]. This population homogeneity makes the popular security solutions high-value targets for security attacks [56, 123]. Any exploit developed for them can render a large user population defenseless, which is highly rewarding for the attacker(s). Furthermore, exploiting security solutions generally grants higher-level privileges, which is a prerequisite for many advanced attacks. While a lot of attention is paid to the detection accuracy and efficiency of security solutions, their own infallibility is generally assumed. This delusive confidence in security solution's integrity can lead to catastrophic failures and compromises. Therefore, tamper-resistance or attack resilience is a very crucial factor in determining the efficacy of a security solution [6, 94].

Recent years have witnessed an increase in malware, often a part of APT attack vectors that are known to incapacitate security monitors first off, leaving the system unprotected. For instance, many incidents of malware attacking host-based intrusion detection systems (IDS) have already been reported [101]. Ensuring the reliability of security monitors in hostile environments is extremely challenging [20]. The kind of malware used to kill anti-virus applications is known as a retrovirus. Section 4.2 takes a detailed look at the recent instances of retroviruses.

The problem of lack of tamper-resistance in security monitors has not gone unnoticed. Many solutions have been proposed to counter it. They vary from weak preventive measures such as stealth-based defense to strong privilege separation models based on virtualization. These solutions are discussed under Section 4.2 where their strengths and shortfalls are analyzed as well. This analysis aids the design of the solution presented in this chapter.

The solution described in this chapter is a watchdog setup based on the principle of *coveillance*. Coveillance involves regular entities watching over their neighbors (for instance, a neighborhood-watch program), unlike surveillance where a central authority watches over other regular entities. Note that most security solutions are based on *surveillance*. An underlying problem of surveillance-based solutions is the difficulty of ensuring that the central authority stays uncorrupted at all times. This gives rise to the classic problem of *'who watches the watcher.'*

The basic architecture of the presented solution consists of a cyclic monitoring topology. Loop architectures and concepts from graph theory have been long used to make architectures robust and some of these are borrowed here in this design [49]. Each participating entity in this solution (referred to as a 'security monitor' in the sequel) is designed to be lightweight. Security monitors perform simple conditional checks on their neighboring security monitors or processes that they are responsible for monitoring. Closed and directed cycles between security monitors ensure that no security monitor remains unmonitored at any time, which makes the monitoring tamper-resistant. This scheme has several benefits such as reducing the size of Trusted Computing Base (TCB), active monitoring and better performance.

In a previous work by Chinchani et al. [26], the effectiveness of a similar scheme in uni-core environment has already been tested out. However, with the widespread adoption of multi-core technology, it becomes necessary to investigate its adoption in multi-core environments as well [42]. Working with multi-cores presents new design and security challenges [95, 74]. Therefore, the solution design for uni-core environment is restructured for its adoption in the multi-core environment [26, 77]. The security strength of the multi-core solution is analyzed against an extensive threat model in an AMD SimNow simulated multi-core environment. Evaluations suggest that the memory overhead incurred by an 8-node (highly-secure) topology is only 0.8%. Furthermore, the time overhead is also a mere 0.3 ns, clearly indicating the efficiency of this solution.

To summarize, this chapter makes the following contributions to the dissertation:

- Focuses on the problem of *'Who watches the watcher?'* and describes in detail the existing solutions for this problem.
- Builds upon the lessons learnt from the analysis of existing solutions and outlines the requirements for an improved solution.
- Describes an extensive threat model encompassing relevant attacks.
- Proposes a solution to ensure tamper-resistance of security monitors in general.
- Evaluates the performance and security strength of the proposed solution in a multi-core environment.

The rest of this chapter is organized as follows – Section 4.2 presents some preliminaries and taxonomy of existing solutions. Section 4.3 lays down the formal requirements for developing an effective solution. Section 4.4 discusses the threat model followed by details of the solution design in Section 4.5. Section 4.6 describes the evaluation and its results. Section 4.7 concludes this chapter with a discussion.

4.2 Preliminaries

This section presents a few preliminaries in order to understand the problem being targeted in this chapter. Existing solutions have been presented as taxonomy for the purpose of better clarity.

4.2.1 Attacks on Security Monitors

In the recent years, numerous malware and attacks have been discovered that primarily target security monitors. Monitors can be subverted either by evasion or by compromise. If successful, an attacker can gain advantages such as unlimited time inside the system and higher system privileges. This enables him to execute more advanced attacks. As mentioned previously, homogeneity in security monitor population results in the popular ones becoming high-value targets.

In a security monitor's design, there is often a trade-off between visibility and isolation. The closer a monitor is to the monitored entity, the more it can observe but the lesser it is shielded from any attacks on the monitored entity. This defines a major difference between host-based and network-based IDSes. Network-based IDS is well isolated from the system that it monitors but cannot watch as closely as a host-based IDS. Thus, host-based monitors are inescapable because they provide a nonpareil view of services running on a host. However, they need to be protected.

One of the first major attacks on host-based security monitors came to light when Phrack published an article on bypassing Tripwire and other integrity checking systems [50]. Tripwire [62] is a tool that monitors files systems for unauthorized or unexpected changes. It maintains a database of signatures/digests of important files (assumed tamper-free). This database is used as baseline for periodically checking the file system for signs of intrusion. The Phrack exploit modified this baseline database to evade Tripwire's monitoring mechanism.

Security monitors can be compromised by either attacking their integrity, confidentiality or availability. Attacking a monitor's integrity involves methods to make it misreport events. This can be achieved by either subverting its data collection (the agent), detection (the director) or alert generation (the notifier) mechanisms. Confidentiality attacks on security monitors are not popular because they usually do not lead to subversion of the entire system. Availability attacks (for instance, switching off the monitor via denial-of-service (DoS) attack) are considered the most effective. Such attacks are well described in the literature [94].

4.2.1.1 Retroviruses

Several sophisticated retroviruses are known to exist and be effective. For instance, Trojan.FakeAV misrepresents the security status of a system. Sometimes it interrupts normal system operations until the user pays for the security fix, downloads and installs it [132]. CoreGuard Antivirus 2009 belongs to the same family and uses social engineering to trick users into uninstalling their security products before installing malware. It achieves this by reporting false security threats [131]. Thinkpoint is a variant of Trojan.FakeAV in circulation since 2010. Websense security lab reported a trojan that installs itself as a Windows input method editor and kills any running antivirus processes [142]. The list goes on [101]. What makes security monitors vulnerable is the vastly diversified set of applications and files that they deal with. This variety leads to the existence of bad code under unpacking and decompression modules [35].

4.2.2 Taxonomy of Tamper-resistant Monitoring Solutions

4.2.2.1 Stealth-based Defense Solutions

Security monitors such as IDS, anti-viruses, etc. typically operate at user level. User-space services are quite vulnerable if attacker gains privileged access to the system [50, 5]. Some systems attempt to hide their security monitors from the attackers. This technique is inspired by the notion of stealth-based defense, *'What the attacker cannot see, he cannot attack.'* For instance, an IDS can be hidden by modifying kernel structures and their presence can be masked using encryption [144]. However, such schemes are easily defeated if the attacker is aware of the stealth mechanism employed [41]. Any solution that relies on the assumption that the attacker has limited knowledge of the system cannot provide strong security guarantees.

4.2.2.2 Replication-based Monitoring Solutions

Surveillance is the generally acceptable paradigm for security monitoring. It is strengthened by employing decentralization of decision-making process, as shown in Fig. 4.1. Replication-based solutions are based on the assumption that compromise of multiple monitors is a lower probability event than compromise of a single monitor. Hence, a majority voting among replicated monitors ensures that the compromised minority is identified and defeated each time. This architecture primarily belongs to the fault-tolerance domain. It can be homogeneous (similar monitors) or heterogeneous (spatially or temporally diverse monitors). Systems such as MAFTIA [111] and Chameleon project [57] use this architecture to make their systems intrusion-tolerant. Replicated architectures provide better security strength than their non-replicated versions. However, if a replicated architecture

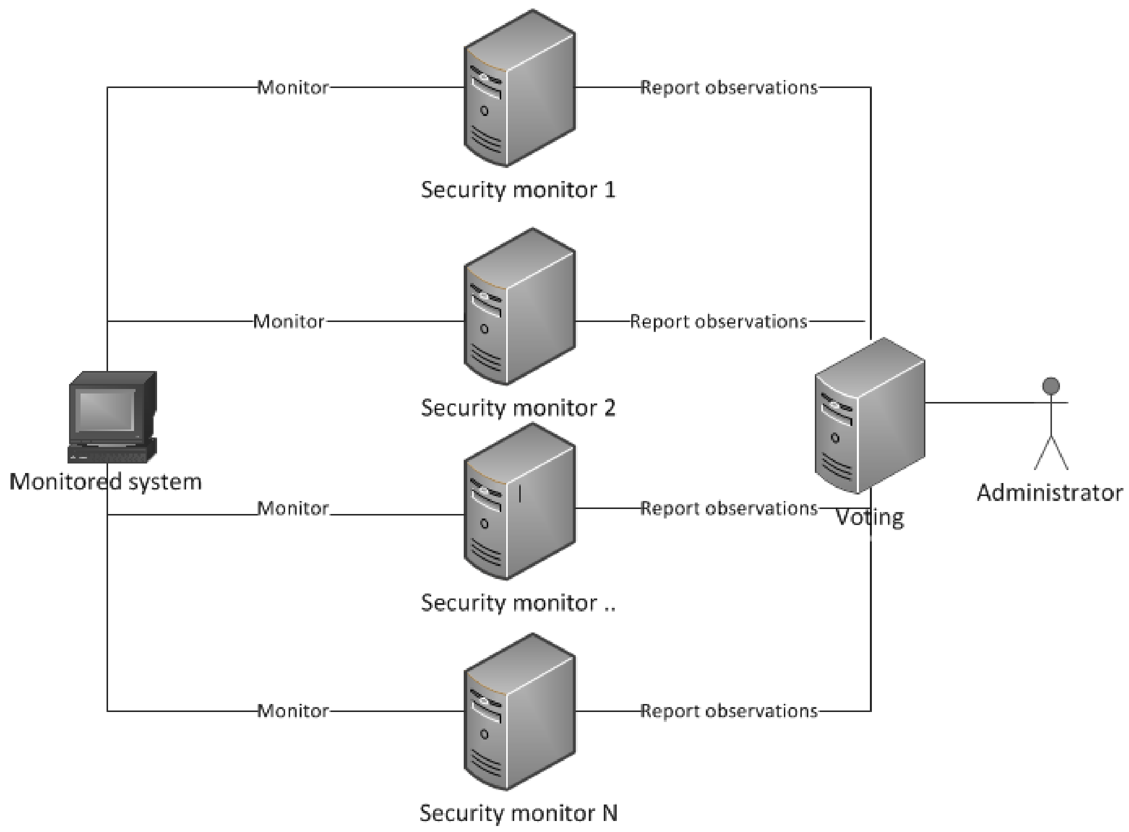

Figure 4.1: Replication-based surveillance solution

is homogeneous and the attacker has compromised one security monitor, it is not unreasonable to assume that the remaining monitors are corruptible as well (especially by automated attacks). Heterogeneous architectures undoubtedly provide better tamper-resistance by introducing diversity to security monitoring replicas. But given enough time and resources, there is a high probability that an attacker can compromise the majority. Thus, replication may be a good security solution for weak adversary model because it depends on assuming that an attacker has limited time, resources or knowledge. However, it may not be as effective against a strong adversary (for instance, insiders) making it a weak security solution. The notion of weak and strong adversary models is discussed in greater details in Section 4.4.

4.2.2.3 Layered Monitoring Solutions

Another popular alternative to replicated architectures is the layered or onion-peel architecture. This architecture involves a chain of security monitors (usually at separate privilege levels) wherein each monitor monitors its predecessor. Refer to Fig. 4.2. This, however, starts a race between the malicious actors to gain the higher privilege. For instance, National Security Agency (NSA) worked with Secure Computing Corporation (SCC) to propose SELinux in 2001 as an attempt to enforce mandatory access control and isolation policies into the Linux operating system [82]. SELinux worked towards facilitating isolated privilege levels across the system. This was a big step towards an effective implementation of the onion-peel architecture. However, the operating system remained susceptible to attacks by the most powerful user on the system. SELinux though provided a good direction for isolating privilege levels but it failed to provide sufficient isolation for co-located processes. The main problem remained that whenever a higher privileged layer is introduced, how to provide protection for this new layer. Solving this problem can lead to infinitely long chains. Another instance of this architecture is IBM's secure hypervisor where a TPM monitors the hypervisor [119]. This TPM can further be monitored by another TPM or hypervisor for strengthened security and so on. The security status of each monitor is transitively propagated to the root.

As before, a heterogeneous variant is more secure than the homogeneous one. It presents several interesting challenges for an attacker. Firstly, the attacker must be able to compromise the various monitors successfully. Secondly, he needs to either shut them down synchronously or in the descending order starting from the root. These attacks require the attacker to have enough time and information (about the order of monitors in the chain and their types). Although this architecture

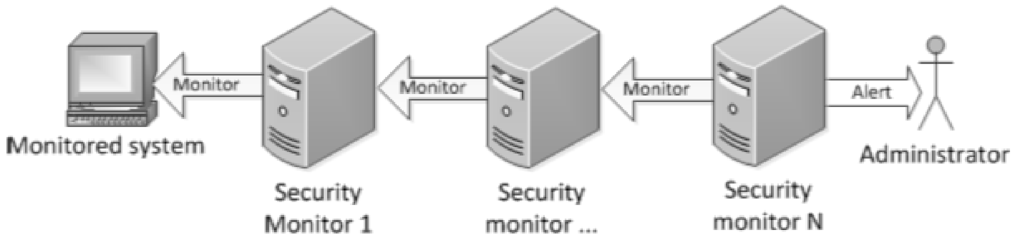

Figure 4.2: Layered or Onion-peel model

provides strong security for a weak adversary model, it falls short of withstanding a strong adversary.

4.2.2.4 Kernel-based Solutions

Realizing the vulnerability of user-space security monitors, security practitioners attempted to deploy security monitors at kernel level [53, 72, 65]. This choice led to an increase in the size of system's TCB which made the system more vulnerable [130].

4.2.2.5 Isolation-based Solutions Using Virtualization

Molina and Cukier [94] favored increasing the level of isolation between user-space security monitors and the monitored entities for better security. One of the popular approaches to achieve this isolation is to move security monitors to a separate virtual machine (VM). This provides security monitors with better threat resistance while maintaining good system visibility [41, 40]. Security monitors in separate VMs benefit from the following three properties:

- *Isolation:* Monitoring entity is isolated from the monitored entity.
- *Inspection:* Monitoring entity can transparently collect data about the monitored entity.

- *Interposition:* Monitoring entity can intercept system calls made by the monitored entity.

Solutions such as Terra [40] and sHype (by IBM) [119] marked the beginning of tool development based on this concept. Terra provides a programming model for running secure and isolated applications with different security requirements side-by-side. This is made possible with the use of a trusted VM monitor that partitions a tamper-resistant hardware into separate VMs. sHype is a hypervisor security architecture for stronger isolation guarantee between the VMs. Laureano et al. [71] proposed a virtual machine monitoring architecture that separates the execution space making the monitor invisible and inaccessible to all the guests/intruders. A data collection module transparently collects system data and sends it to the VM running the monitor. An alert is generated when an intrusion is detected. This alert can take preventive measures such as blocking system calls by the suspected guest user.

Another important host-based monitoring system is application-level firewall. It is responsible for tracking network traffic back to their source processes and comparing it to the existing access policies to sieve out illegitimate traffic. Network and host level firewalls do coarse-grain filtering by relying on port and IP address based access policies only. Therefore, for better visibility, these firewalls need to be able to collect operating system level information about the processes. VMWall [129] is such a Xen-based solution that implements a firewall to detect bots, worms, backdoors, etc. It correlates network traffic with the information gathered via VM introspection. Introspection is the process of inspecting software inside one virtual machine from another virtual machine. VMWall runs on a trusted VM (dom0) and singles out the processes in the monitored VM associated with any suspicious traffic flows. Data structures necessary for VMwall are secured with existing kernel integrity protection mechanisms. VMwall can also successfully block backdoors, bots and worm traffic emanating from a protected system.

All introspection-based tools have a common drawback, viz. passive monitoring. Passive monitoring is when security tools monitor via external scanning and polling and as a result, cannot guarantee interposition of events. Active monitoring, on the other hand, places a hook inside the system being monitored. When execution reaches the hook, the control is passed on to the security tool.

File-system integrity tools are effective but do not monitor files in real-time. This opens the possibility of time-of-check-to-time-of-use (TOCTOU)-like attack which is elaborated in Section 4.4. XenFIT [114, 14] was proposed as a real-time alternative based on Xen virtual machines. XenFIT did away with the requirement of creating and updating the baseline database which has always been the Achilles' heel of file-system integrity tools. XenFIT claims to be tamper-resistant and stealthy even if an attacker gains privileged access to the VM. It performs real-time monitoring by strategically placing hooks/breakpoints in the kernel code of Xen (active monitoring). When these breakpoints are hit, control is transferred to XenFIT which then collects all the required activity related data at that point in time. It basically works like a kernel debugger. Because it runs in the user-space of Dom0, it is invisible and inaccessible to the processes running in DomU. This feature provides it with good tamper-resistance. It is easy to deploy and collects far richer information than traditional file-system integrity tools.

Lares [107] provides secure active monitoring using platform virtualization. It places hooks inside the untrusted VM so introspection of events can be guaranteed. The monitored VM contains *trampoline* code that communicates with Lares in Dom0. Trampoline helps Lares implement memory access policies in the guest

VM. The trampoline functionality is self-contained (does not depend on kernel functions), non-persistent (does not require data which was generated during previous hook activations) and atomic. Hooks are secured with a memory protection mechanism. When a guest VM requires a write change in a certain memory page, the hypervisor checks whether the requested memory address is write-protected. If so, the change is not allowed and the required change from the guest VM is not propagated into the actual physical memory. A list of protected memory regions is stored and maintained by a Lares component.

Flicker [87] uses hardware-supported late launch and attestation for integrity measurement. It is vulnerable to scrubbing attacks since the monitored system is responsible for invoking Flickers integrity measurement capability. HyperGuard [145] is the first system management mode (SMM)-based framework that attempted to provide tamper-resistant integrity measurement of hypervisors. It utilizes a hardware timer to periodically trigger system management interrupt (SMI). Once triggered, the SMM code checks the hash of the most privileged software running in the protected mode. HyperCheck [140] uses a peripheral component interconnect (PCI) network card to periodically trigger SMI. This is to ensure that unlike in HyperGuard, SMI cannot be triggered by the adversary. Both these tools are vulnerable to scrubbing attacks since they cannot invoke integrity measurement without alerting the monitored hypervisor. As in the cases of Hyperguard and HyperCheck, HyperSentry [10] also utilizes SMM to monitor integrity of hypervisors. It uses intelligent platform interface and baseboard management controller of servers to trigger SMI periodically. Wang et al. [138] demonstrated a new class of attacks that can defeat all SMM-based periodic integrity checking mechanisms.

4.2.3 Limitations

As researchers try to design better tamper-resistant security monitors, they add more complexity to existing solutions leading to further exploits. Any increase in security usually comes with increased overhead and new hardware dependencies. Note that scalability of security solutions is very important as well. Employing virtualization-based solutions for an expanding system can quickly add to the overall overhead. Use of hypervisors and SMMs may also make a system vulnerable by providing hiding places for rootkits. Many researchers have talked about such hypervisor and SMM-based rootkits [67, 63, 34].

4.3 Formal Requirements

In addition to providing tamper-resistant monitoring to security solutions, an ideal solution must satisfy the following additional requirements (derived from the discussions in Section 4.2):

- *Easy verifiability:* Any solution should aim to avoid unnecessary complexity as it further leads to security issues. Simplicity in design and implementation is the key to designing a good tamper-resistant solution. If the solution is simple, it can be easily verified to be free of any loopholes.

- *Low time and performance overhead:* An effective solution should incur a low time and performance overhead. Low time overhead is especially important for mission-survivable systems. Low performance overhead is good for the scalability and performance of the overall system.

- *Address the threat model:* It is essential that the solution attempts to address as many threats as described in Section 4.4.

- *Absence of window of vulnerability (WoV):* A WoV (a delay between an event and its notification) can jeopardize the integrity of a security monitor or the entire system. For instance, TOCTOU attacks. A good solution should avoid their creation especially in the design and deployment phases.

- *Size of trusted computing base (TCB):* It is easy to assume that a security solution (or at least some part of it) belongs to the TCB. This makes the job easier for solution designers. However, the TCB size should always be kept small for easy verifiability. When TCB becomes large or its components are updated often, it becomes impossible to ensure its integrity. This further leads to serious security issues.

- *Strong security guarantees:* A good solution should provide strong security and tamper-resistance guarantees; meaning that the solution should work well for strong adversary model (described in Section 4.4). Attackers should not be assumed to have permanent limitations such as lack of knowledge about the system or limited time or resources.

4.4 Threat Modeling

4.4.1 Adversary models

The amount of damage that an adversary can inflict on a system depends upon the time spent, resources employed and his knowledge of the system. It is impossible to model a case-by-case adversary behavior based on these factors. Therefore, two models that mark the upper and lower boundary cases of attackers' capabilities are considered here [26].

- *Weak Adversary Model:* This model describes adversaries with limited time, resources and knowledge about the target system. Advanced targeted attacks on a system usually require abundance of at least one of these. Therefore, this model generally covers non-targeted attacks and closely resembles the behavior of script kiddies, generic viruses, worms, etc. In fact, the most commonly occurring attacks on the Internet can be classified under this category. Whether such attacks are successful or not, is probabilistic from an adversarial point of view. Existing system vulnerabilities play a major role in defining their success. Thus, the actual probability of compromise by a weak adversary depends on the attacker's capabilities along with several other factors such as the number of vulnerabilities per 1000 lines of code, frequency with which patches are applied and other security hardening measures at the system's side. Any security solution developed for this adversary model provides weak security guarantees.

- *Strong Adversary Model:* This model describes powerful and persistent adversaries with significant time, resources and knowledge about the target. Knowledge can be pre-existent or can be gathered by means such as multi-stage reconnaissance attacks or social engineering. Adversaries under this category usually employ advanced attack strategies and stealth. Thus, they are capable of executing targeted attacks. For instance, consider the insiders who can use their familiarity and proximity to the mainframe infrastructure to launch potentially devastating attacks. Any security model developed based on this adversary model provides strong security guarantees.

4.4.2 Attack Strategies

Attack strategies described below are useful for both (weak and strong) adversary models. However, the success probability of an attack depends on the adversary type and techniques.

- *Exploiting population homogeneity:* Attacking a widely popular security monitor can be extremely rewarding for an attacker. Attackers have easy access to such monitors for experimentation and one exploit is enough to affect a large user population. Such attacks usually require lesser resources and time than targeted attacks. Hence, this attack strategy is usually categorized under the weak adversary model. Its possible categorization under strong adversary model depends on the attack vectors and goals [17, 6, 35].

- *Multi-shot attacks and target modeling:* This attack strategy involves several stages such as reconnaissance, decision-making and planning. These stages are aimed at thoroughly understanding a target. Because such an attack requires ample resources and time, this strategy is more suited for classification under strong adversary model. Stealth is generally an important aspect of such attacks.

4.4.3 Attack Patterns

Following are some specific attack patterns and instances that threaten the integrity of security monitors:

- *Planting false data:* This is a passive attack that waits for security monitor to read-in malicious or malformed input from the infected machine or process. For instance, Microsoft's security feature in 2001 (based on StackGuard) was designed to safeguard gcc compilers against buffer overflow attacks. Ironically, it in itself contained a buffer overflow vulnerability that could be exploited by supplying malformed input [51, 21]. An effective solution to this problem is to extensively stress test security monitors before release. Monitors should not trust the monitored system/process to provide clean input. Consequently, any input coming from a monitored system/process must be strictly validated.

- *Fake alerts:* Generating fake alerts on a system can comprise a social engineering attack. Alerts can prompt administrators to perform tasks beneficial to an attacker. For instance, Trojan.FakeAV, Thinkpoint, etc. A solution is to provide an authenticated interface between alert generation module and the recipient such as access controlled log files. If the administrator sees an alert, he can check the log file to confirm the alert's authenticity.
- *Attacks on checking and comparison algorithms:* This attack involves compromising components of a security monitor that are generally assumed uncompromisable. This assumption is generally made because of a high privilege requirement or because security monitors are assumed to be tamper-resistant. If executed successfully, such attacks can take down the entire monitoring mechanism. In this chapter, we move past the assumption that security monitors are tamper-resistant. Therefore, this attack pattern, though ignored by many, is dealt with by our solution.
- *Exploit:* An exploit occurs when an adversary is able to attack the target and take complete control of it (typically in terms of a root-shell). Worse still, he may be able to install a root-kit to replace the security monitor with a fake one that serves his own purpose.

- *Crash attacks:* Crash attacks are the most common attack patterns and are relatively simpler to create. Usually, system crashes are considered an acceptable alternative by system administrators when compared to a system hijack. However, crashing security monitors is a serious issue as it may eventually lead to system hijack. Crash attacks are generally considered to be easily detectable. But crashing a security monitor or replacing it with a fake one (an exploit) may not be easily noticeable. Crash attacks can be executed in any number of ways such as planting false data, DoS attack via system overload or social engineering. The solution presented in this chapter deals with such attacks by monitoring all security monitors for their continuous operation.

- *TOCTOU attacks:* Such attacks occur when a security system is fooled into thinking that there is no security violation during the check on an object but the violation actually occurs during the use of the object. A monitor gathers information via a sensor or probe. Typically there is a lag between the time that the information is gathered and the time it is processed to potentially raise an alert. Attacks happen during this lag. Refer to Fig. 4.3.

  Imagine an event notification system that has a delay between event occurrence and its notification. During this window, an attacker can compromise the security monitor and disable the notification. The administrator will never be alerted even if the monitor or the entire system has been compromised.

  TOCTOU attacks are especially relevant to multi-core architectures. They stem from attacks such as memory hogging. Memory hogging is a kind of DoS attack where one core consumes shared memory unfairly [95]. This

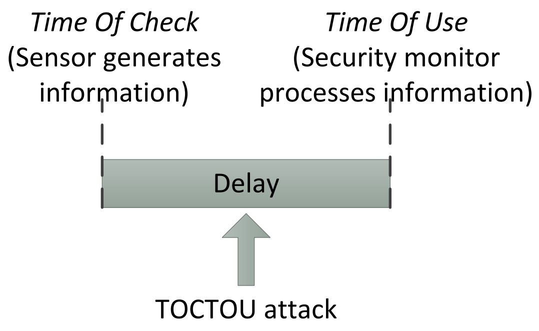
Time Of Check
(Sensor generates information)
Time Of Use
(Security monitor processes information)
Delay
TOCTOU attack

Figure 4.3: Time of check to time of use (TOCTOU) attack

results in performance degradation at other cores due to resource scarcity. It further leads to operational delays resulting in TOCTOU attacks. This vulnerability can be handled via conditional checks. A security monitor can raise an alert if it observes exceptionally high scheduling delays affecting a monitored process.

- *Scrubbing attacks:* These attacks occur when a security monitor is invoked periodically via some sort of signaling in order to observe a process. If the attacker has enough privileges to observe these signals, he can scrub his traces when it happens. Consequently, the monitor notices nothing out of the ordinary and returns the control to compromised process. These attacks are present in SMM-based security monitors that require invocation. For instance, HyperCheck [140], HyperGuard [145], etc.

4.5 Solution Design

The work presented in this chapter is an extension of the work by Chinchani et al. [26]. All the design decisions and innovations hereafter are guided by formal

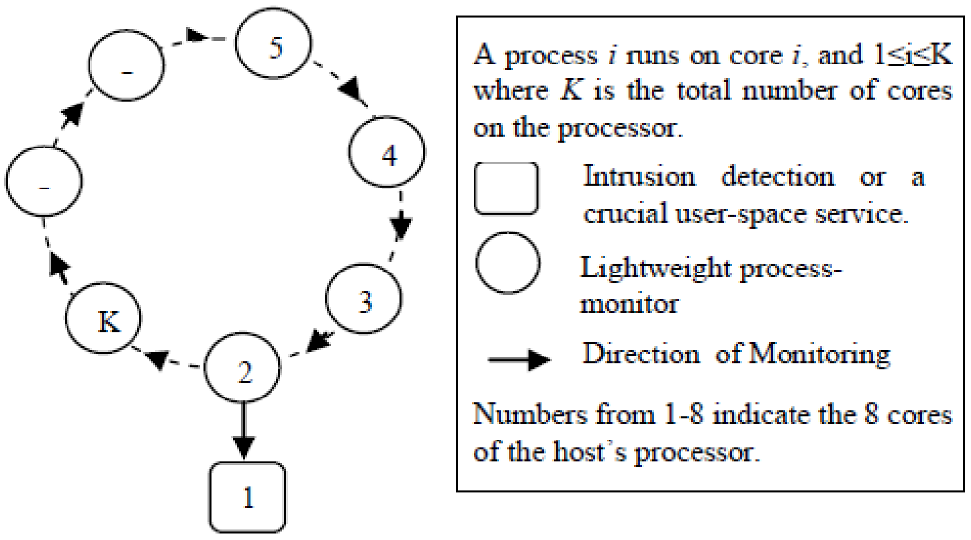
5
4
3
2
1
K
-
-
A process *i* runs on core *i*, and 1≤i≤K where *K* is the total number of cores on the processor.
Intrusion detection or a crucial user-space service.
Lightweight process-monitor
Direction of Monitoring
Numbers from 1-8 indicate the 8 cores of the host's processor.

Figure 4.4: Simple cyclic arrangement of security monitors

requirements described in Section 4.3.

As discussed previously, the challenge is to always have a security monitor monitored for tampering. Surveillance-based solutions trying to provide tamper-resistance guarantees usually lead to infinitely long chains of monitors. The same effect can be achieved via a cyclic topology. Cyclic graphs have no open ends which means that if security monitors are arranged in a directed cycle, they will all be monitored at all times. Refer to Fig. 4.4.

Below is the description of how formal requirements described in Section 4.3 shape the solution presented in this chapter:

- *Easy verifiability:* This requirement calls for a simple design whose security strength can be easily verified. It was inferred from the survey in Section 4.2 that solutions built upon virtualization tend to get complex and cumbersome and hence, virtualization is avoided in the design. Instead of following the suit of current surveillance-based monitoring architectures, inspiration is derived from coveillance. This choice made the solution design tremendously simple. The presented design is a structural arrangement of lightweight processes that monitor each other in a cyclic fashion. These processes are intentionally simple and replicated so they are easy to update and verify. Additionally, the structural arrangement is based on concepts derived from graph theory making it easier to understand its security properties and any possible loopholes that may ensue.

- *Low time and performance overhead:* The presented solution is composed of lightweight processes and leverages the parallelism offered by multi-core architectures. To further gain performance, process communication or event reporting takes place over the kqueue subsystem. Kqueue subsystem is a kernel level communication mechanism which is extremely efficient and lightweight. As a result of these design and implementation considerations, the evaluations in Section 4.6 report low time and performance overheads.

- *Address the threat model:* While designing the solution, all the threats from Section 4.4 have been addressed. It will be apparent as more design details are discussed later on.

- *Absence of WoVs:* Creation of any WoVs is avoided by employing a cyclic architecture and the kqueue subsystem. Cyclic architecture ensures that no participating entity is unwatched at any time and kqueue subsystem ensures that events are reported without any delay. Section 4.6 specifically discusses the handling of TOCTOU and similar attacks.

- *Small size of the TCB:* One of the most important features of the presented solution is that it does not impact the existing TCB. Because all participating entities are being watched by each other in a cyclic fashion, no single entity needs to be assumed secured, unlike in surveillance-based architectures. However, the design leverages existing TCB components such as the kernel level communication mechanism in order to ensure a no-delay and tamper-resistant event reporting.

- *Strong security guarantees:* The solution is designed to withstand a strong adversary model. Thus, the attacker is not bounded by limited time, resources or knowledge. Unlike the replication-based or layered monitoring solutions, knowing the entities or their arrangement order does not weaken the security strength of this solution.

In its simplest form, the solution consists of several user-space security monitors. These security monitors can be on the same machine or on separate real or virtual machines. The solution proposed here can be used with almost all existing security monitoring schemes to strengthen their tamper-resistance. Figure 4.4 presents a simple cyclic topology of this solution. The nodes are the security monitors and the directed edges represent the monitoring relationships between them. For instance, in this case, security monitor 3 is monitoring security monitor 2 and 4 is monitoring 3. Note that no monitor is left unmonitored. One of these security monitors (called 'the primary' in the sequel) is designated to monitor critical processes (for instance, an IDS) on the system. The primary security monitor is in a cycle with other security monitors (called 'the watchers' in the sequel) that helps ensure its tamper-resistance. Because watchers are only required to monitor other watchers (including the primary), they can be lightweight processes with simple functionality. This makes it possible for the solution to provide strong

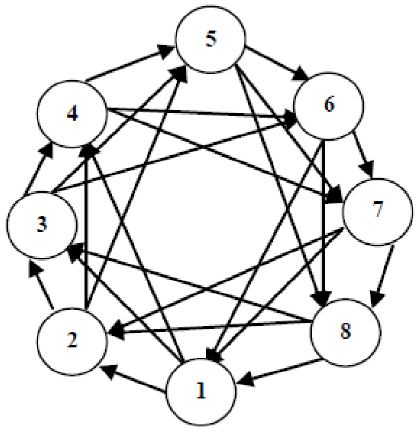

Figure 4.5: Circulant digraph with number of nodes=8, degree as incidence=3 and jumps=1,2

tamper-resistance without any design and implementation complexities.

Building upon this idea, a generic coveillance-inspired solution based on circulant digraphs is presented here. A circulant digraph $C_K(a_1, a_2, ..., a_n)$ with K vertices $v_0, v_1, ..., v_{K-1}$ and jumps $a_1, a_2, ..., a_n$, where $0 < a_i < \lfloor K/2 \rfloor$, is a directed graph such that there is a directed edge each from all the vertices $v_j \pm a_i mod K$, for $1 < i < n$ to the vertex $v_j, 0 < j < K - 1$. It is also homogeneous i.e., every vertex has the same degree (number of incident edges), which is $2n$, except when $a_i = K/2$ for some i, when the degree is $2n - 1$. Figure 4.5 presents a directed circulant digraph with 8 nodes, degree of incidence 3 and jumps $1, 2$.

4.5.1 Topologies

A monitoring solution, cyclic or not, can have numerous topologies. In this case, topologies vary from a simple cycle to a topology with multiple degree of incidence. A few basic topologies with strong tamper-resistance are discussed here. In order to compare these topologies, one may ask the following two questions:

- *How secure a topology is?*
- *How efficient a topology is?*

These questions form the basis of the discussion and evaluation of these topologies.

- *Simple Ring or simple cycle.* Simple ring topology represents an ordered cycle of watchers as shown in Fig. 4.4. It offers a much lower probability of subversion compared to the onion-peel model [26]. If an attacker does not know about the order of watchers in the simple ring, he needs to try $(n! - 1)$ permutations (worst case) before he finds the right order. Note that alerts will be raised for all these attempts indicating an attack pattern and sent to the security administrator. In order to deal with a strong adversary model, this topology considers insider attacks as well. Even if the attacker (may be an insider) knows the arrangement of monitors in the ring, he needs to synchronously compromise watchers all at once. There are ways to prevent this from happening as discussed later.

- *Circular digraph.* Circulant digraph is characterized by a higher degree of incidence (more watchers watching). This further reduces the probability of a watcher being left unmonitored at any time. Refer to Fig. 4.5.

  Simple ring topology is a special case of circulant digraph topology when degree of incidence is 1. However, a circulant graph topology (with degree of incidence > 1) is much more secure than a simple ring topology. This is because the expected number of attempts required to find the right permutation increases exponentially because the attacker does not know the degree of incidence, the jump and the order of watchers.

- *Adaptive ring.* Since raising a large number of alarms is counter-productive to system performance, a circulant topology though effective is not optimal. Even if an attacker is not in a position to attack, he can tamper with the deployed solution to make it raise a large number of useless alerts. In intrusion prevention systems (IPS), each alert results in a series of defensive reactions. Thus, a large number of alerts can potentially lead to a DoS attack. To counter this threat and reduce the number of alerts produced by a circulant topology, an adaptive topology is proposed. It predicts system load and tries to maintain cyclic monitoring at all times. This is especially useful for multi-core systems. It requires that watchers at each core track the load on other cores. As shown in Fig. 4.6, the initial state of this topology is set to be a simple ring. If the watcher on core 2 realizes that core K has just been assigned a lot of new monitoring tasks, it starts monitoring watchers K and K-1, both. Similarly, if core 2 gets heavily loaded, watcher 3 starts monitoring watchers 2, K and K-1. So, the cores that are lightly loaded take up the additional responsibility of monitoring the watchers on heavily loaded cores and their respective assignments.

  Formally, an adaptive topology can be represented as a directed graph $G = (V, E)$ where,

  - V is the set of all vertices in graph G. In other words, it is the set of all watchers.
  - E is the set of ordered pair of vertices of V. A directed edge from v' to v'' is represented as $v' \rightarrow v''$ where, $v' \in V$ and $v'' \in V$.

  For convenience, G is represented as an adjacency matrix $A = a_{ij}$ of size $|V| \times |V|$ such that,

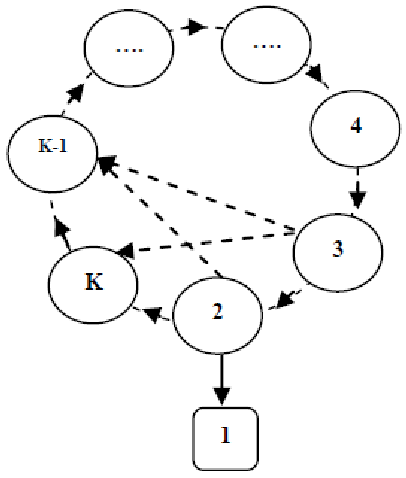

Figure 4.6: Adaptive topology when cores 2 and K are heavily loaded

$$a_{ij} = \begin{cases} 1, & \text{if a directed edge } v_i \rightarrow v_j \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

The following two conditions must hold true at all times:

- $\sum_{i=1}^{|V|} a_{ij} \geq 1, \forall j$ where, $1 \leq j \leq |V|$. If this condition is violated for any j, vertex v_j cannot be considered secure anymore and is eliminated from the chain of trust.
- At least one cycle must exist in the graph containing all vertices (watchers) in G except the ones eliminated due to violation of the first condition.

The solution does not allow loops of length 1 in A because it represents a condition where the monitor monitors itself. Therefore, $a_{ij} = 0$ where $i = j$. In other words, all diagonal elements in A are zero.

Another set $L = l_m$ of size $|V|$ is defined, such that l_m represents the load condition of each vertex in V where, $1 \leq m \leq |V|$. This represents the load on the core where each watcher from V is running. In other words, L represents the delay conditions of watchers in V. The load threshold is represented as $\hat{l}$.

As described earlier, $a_{ij} = 1$ for any $1 \leq i, j \leq |V|$ if a directed edge $v_i \rightarrow v_j$ exists. The edges in this graph are reassigned based on the load on each vertex as represented in L. Therefore, for every a_{ij} if $l_i \geq \hat{l}$ then the need is to reassign the monitoring of some of v_j's where, $v_i \rightarrow v_j$ exists, to the monitors of v_i. Therefore, as a best effort *without violating the two basic conditions*, an x is picked where $a_{xi} = 1$ and $a_{xj} = 1$ is arranged such that j satisfies the condition $a_{ij} = 1$. The eventual target is to obtain $l(i) \leq \hat{(l)}\ \forall i$ where, $1 \leq i \leq |V|$.

The probability of subversion for adaptive cycle topology is equal to the probability of subversion for circulant digraph topology. However, the number of attempts required to find the right order of watchers is much larger than in circulant digraph topology (in the worst case). This is because the degree of incidence and jumps are always changing dynamically. Therefore, an adaptive topology provides better performance and stronger tamper-resistance as compared to the circulant digraph topology.

4.6 Implementation

4.6.1 Monitoring

Monitoring among watchers can be implemented in several ways:

- *Periodic checks:* An elementary approach is to have processes periodically ping their watchers to signal their liveness (mechanism called heartbeats) or the watchers can periodically poll their assignments. This solution though simple does not fit the purpose. The reason being that it is difficult to choose a heartbeat/polling interval that satisfies the requirements described in Section 4.3. If the interval is too small, it results in a large overhead. If it is too large, there can be a substantial delay between the event (watcher's assignment being killed) and its notification (watcher realizing it). This will lead to TOCTOU-attacks if killing watchers is carefully timed (for instance, at the beginning of each heartbeat interval).

- *Direct closed-loop monitoring of watchers using system calls:* Support for direct monitoring of processes is available in most *NIX operating systems in the form of ptrace(2) family of system calls [55]. The proc filesystem provides an alternate mechanism for system call tracing called *Truss*, which is a program in FreeBSD. The challenge with this approach is its closed-loop aspect. When watchers are arranged in a closed-loop, taking control of the traced watchers on each event results in the possibility of an exponential cascade of events and deadlocks [39]. Design decisions at the operating system level occlude such closed-loop relationships between traced watchers. Refer to Fig. 4.7.

- *Non-repudiable event generation and delivery system:* FreeBSD provides a good kernel (asynchronous) event notification mechanism called *kqueue* [75]. Kqueue is highly scalable and proposed as a replacement for poll(2) and select(2). Web servers have been reported to achieve significant performance gains when using kqueue subsystem for socket events.

  Kqueue enables watchers to listen for the events generated by their assignments. Refer to Fig. 4.8. It first creates a kevent handle similar to the file

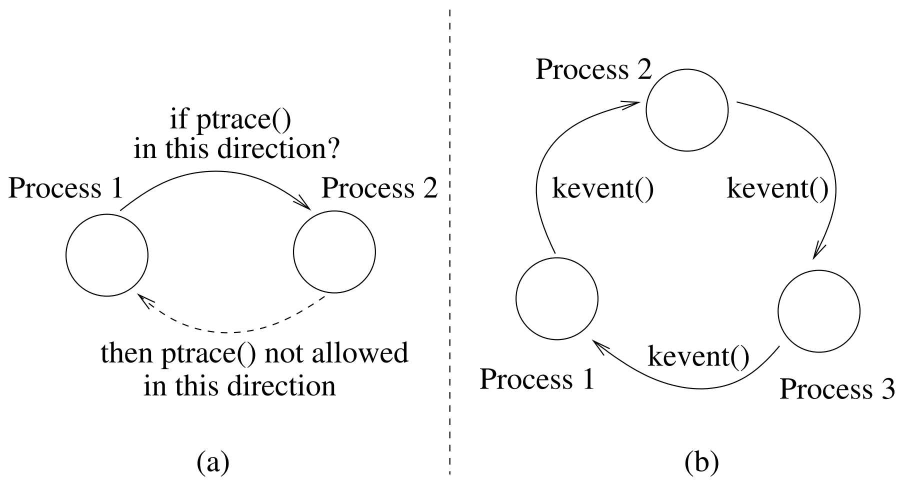

Figure 4.7: Monitoring watchers using ptrace(2) and kevent(2) (a) ptrace(2) does not allow a loop; (b) kevent(2) being asynchronous allows such loops

descriptor returned via an open(2) system call. Then, it registers a listener function for each of its assignment. The listener is essentially an idle event loop triggered by the events when they occur. The kqueue subsystem allows multiple watchers to register listeners for the same event. Since the event delivery and notification mechanism is asynchronous unlike in ptrace(2), it is possible to implement closed loops among the monitors (see Fig. 4.4). Although a user-space process generates a system call, the corresponding event is created and processed inside the kernel. Therefore, this event cannot be repudiated or falsified.

4.6.2 Experimental Setup

Companies such as Intel and AMD have made significant progress in multi-core technology. Clearspeeds CSX600 processor with 96 cores and Intels Teraflops Research chip with 80 cores are the latest in this line [100, 54]. However, such systems do not have a strong presence in the commercial market yet. This generally re-

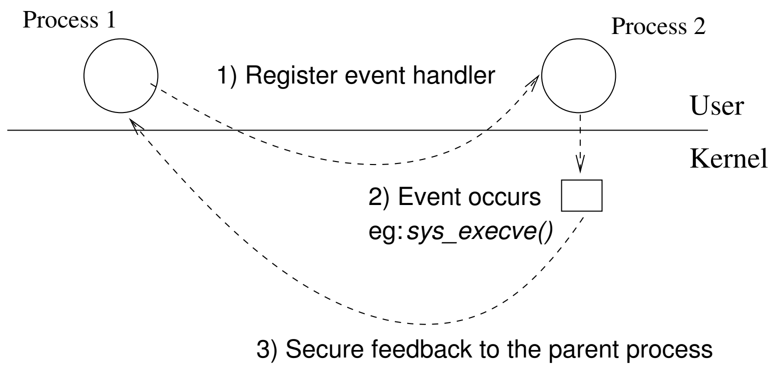
Process 1
Process 2
1) Register event handler
User
Kernel
2) Event occurs
eg: *sys_execve()*
3) Secure feedback to the parent process

Figure 4.8: Watcher monitoring using kqueue subsystem

stricts researchers to use a small number of 2-6 cores. In order to bridge this gap between unavailability of present technology and researching the future trends, multi-core simulators have been developed [92, 136]. These simulators emulate the functioning of a multi-core platform on a system with lesser number of cores (even on uni-core systems). AMD SimNow is one such open-source multi-core simulator that closely emulates the non-uniform memory access (NUMA) architecture. It is used here as a test-bed to experiment with simple ring and circulant digraph topologies.

The following experiments are conducted on Intel Pentium Core2Duo 2.1 Ghz processor with 4GB RAM. AMD SimNow is installed on Ubuntu 10.04 (the host operating system). Inside AMD SimNow, FreeBSD 7.3 is installed as a guest operating system. All experiments run on this guest operating system. This system is configured to use emulated hardware of AMD Awesim 800Mhz 8-core processor with 1024 MB RAM. Kqueue is used as the event delivery/notification subsystem.

4.6.3 Efficacy Against Attacks

The two primary performance metrics in a multicore system are time and memory overheads. The setup time is defined as the time taken for the kqueue subsystem

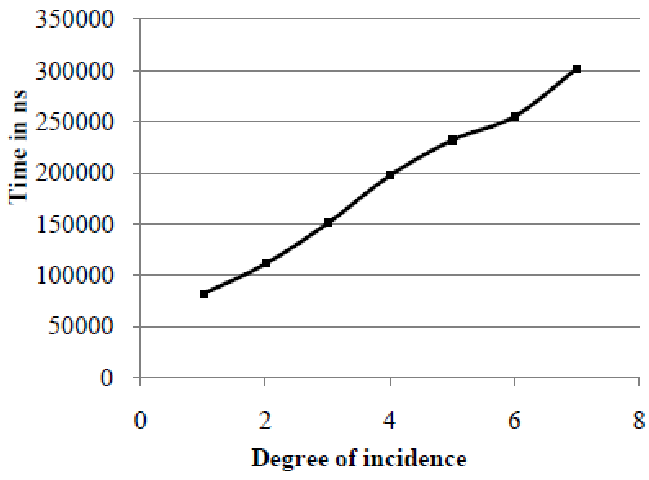

Figure 4.9: Setup overhead for circulant digraph topology with 8 watchers

Table 4.1: Categorization of circular digraph topologies

Configuration	Number of watchers	Degree of Incidence
Series 1	2	1
Series 2	3	1,2
Series 3	4	1,2,3
Series 4	5	1,2,3,4
Series 5	6	1,2,3,4,5
Series 6	7	1,2,3,4,5,6
Series 7	8	1,2,3,4,5,6,7

to load. This is the only major time delay that this solution incurs. As shown in Fig. 4.9, setup time increases linearly with increase in the degree of incidence. With 8 watchers in a circulant digraph topology and maximum degree of incidence (i.e., 7), the worst case setup delay is observed to be 0.3 ns.

The set of cases to be studied involves different circulant digraph configurations with varying number of watchers and degrees of incidence as shown in Table 4.1. For these cases, jumps start from a minimum of 1 and are incremented by 1 until they satisfy the degree of incidence.

The following attack scenarios are used to test the security strength of this

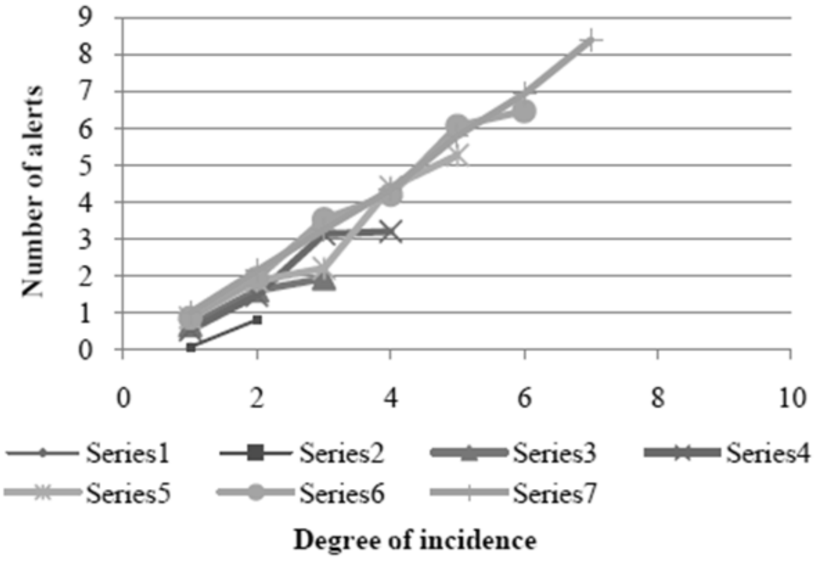

Figure 4.10: Average number of alerts generated for killing watchers in sequential order without delay, under light system load

solution.

- *Experiment 1: Killing watchers without delay (under light system load):* The worst case scenario is where the attacker already knows the correct order of watchers in the cycle. In Fig. 4.10, the number of alerts generated shows the sensitivity of this solution towards the crash attacks via SIGKILL.

- *Experiment 2: Killing watchers without delay (under heavy system load):* Experiment 1 was repeated under heavy load conditions to determine the impact of increasing system load on frameworks sensitivity (number of alerts generated) to an attack. A heavy load condition is simulated by running OpenSSL benchmark in the background. In this emulated multi-core environment, a maximum of 6,164 processes can run on FreeBSD operating system. Six thousand processes were deployed to achieve nearly 100% CPU consumption for all cores. As seen in Fig. 4.11, the number of alerts generated is lesser than in Experiment 1. This is because now watchers have to

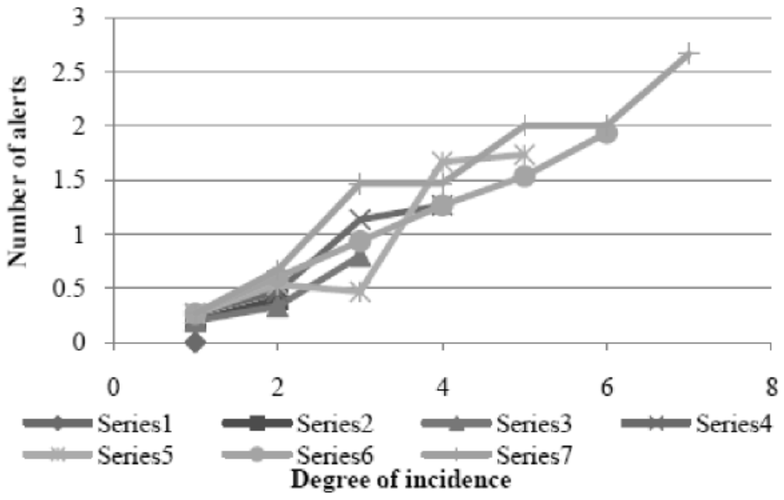

Figure 4.11: Average number of alerts generated for killing watchers in sequential order without delay, under heavy system load

wait in the scheduling queue longer than in Experiment 1. Thus, to ensure that we get at least one alert under all conditions, there should at least be 3 watchers in the configuration.

4.6.4 System Overhead

Memory overhead is defined as the amount of memory consumed by a running instance of the solution as a percentage of the entire system's memory capacity. As shown in Fig. 4.12, it increases linearly with the degree of incidence. A circulant topology with 8 watchers and degree of incidence 7 incurs a 0.8% memory overhead.

4.7 Discussion

This chapter presents a solution to the problem of ascertaining tamper-resistance of security monitoring schemes in the user-space. Note that the security monitors

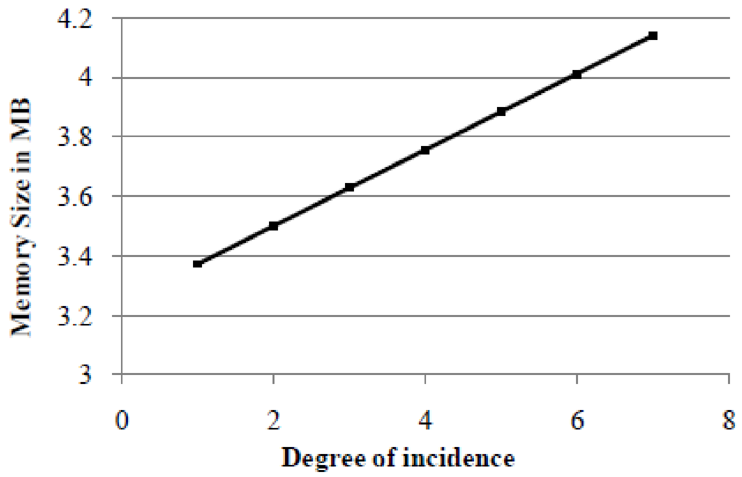

Figure 4.12: Memory overhead for a watcher in an 8-node circulant digraph topology

in kernel space are not considered here. Such monitors are though more secure than the ones in the user-space but they introduce significant problems to the trusted computing base, making it vulnerable. Therefore, security monitors in the user-space are a safer alternative if their tamper-resistance can be assured. This is what the chapter focuses on. There exist several solutions in the literature for this purpose. Therefore, the literature is reviewed to understand the limitations of existing solutions, as well as, derive important lessons. Based on these, a new solution is designed. It consists of lightweight watchers arranged in a cyclic configuration. To make it more generic and effective, circulant digraph and adaptive topologies are discussed. Circulant digraph topology provides good security. Adaptive topology aims to provide a fine balance between security and performance overhead.

The quality of tamper-resistance provided by this solution is implementation-dependent to some extent. This is because the directed edges in the solution design represent periodic checks for specific conditions, such as a check if the monitored program is continuously running. The conditions defining *unexpected behavior* are decided and implemented by security practitioners. Hence, the term *tamper-resistant* is preferred over *tamper-proof* to describe this solution. The watchers are expected to be lightweight allowing easy formal verification of code correctness. The solution is expected to have a low number of false positives because of the simplicity of checks. Any false negatives will most likely result from the developers missing to implement all the conditional checks.

Due to the popularity and widespread adoption of multi-core systems, this solution is evaluated in a multi-core environment. Note that it works effectively in a unicore environment as well [26]. For evaluation, the crash (kill) commands and exploits are executed in sequence without any user delay. This is to simulate a strong adversary model where the attacker has time to develop targeted and automated attack scripts. The most effective attack against this solution is the crash attack by an adversary with escalated privilege. If he can issue a killall command for all the watchers, the solution can be subverted. However, most *NIX operating systems allow killall to send SIGKILL signal to processes belonging to the same group. In order to increase framework's resistance to such crash attacks, alternate watchers can be organized under two different groups. Watchers with even PIDs (process IDs) retain their default GID, which is the PID of the parent process. The GID of watchers with odd PIDs is changed to their respective PIDs. Now, a SIGKILL signal sent to the default GID of the group will successfully kill the watchers with even PIDs but the odd ones will still raise alerts. Killing multiple groups involves issuing multiple kill commands which introduces delay that will lead to alert generation.

To summarize, the solution presented in this chapter protects security monitors against APTs by providing them with strong tamper-resistance. In addition to that, it is lightweight in terms of time and performance overhead and thus, can be employed as part of a mission survivable solution.

In the next chapter, we discuss the design of SWAT in a distributed environment and how the solution presented in this chapter applies to it.

Chapter 5

Distributed Architecture for Survivability

5.1 Introduction

Chapter 3 described an advanced persistent threat (APT)-based attack model and a centralized version of SWAT to withstand it. Its concluding discussion identified two major tasks that needed further work. First was to ensure that the security monitors at each host are tamper-resistant. In this regard, a solution was presented in Chapter 4. Second was to improve the hardware-based capabilities for a secure and surreptitious node-to-node verification. This chapter proposes the possible use of trusted platform module (TPM) for this purpose. In addition to it, centralized SWAT is extended and restructured to work efficiently and effectively in a distributed environment.

As discussed earlier, advanced adversaries are capable of maintaining multiple footholds inside the system. Detecting and recovering a subset of these footholds may raise red flags for the intruders, leading them to refine their attack strategies.

For instance, they may execute contingency scrubbing attacks and erase their traces by either deleting files or corrupting certain system states, thereby foiling the mission. This kind of behavior works against mission survivability requirements. Therefore, any mission survivable solution must be tamper-resistant and should not divulge any information prior to considering and analyzing all the repercussions.

Similar to the centralized SWAT described in Chapter 3, the solution presented here leverages concepts of deception (hiding) in a hardware-based security setup [146, 28]. It does not immediately raise an alert or activate recovery procedures on intrusion detection. Instead, it continues to behave normally and observe in order to better understand attacker intent, objectives and strategies (AIOS). This approach has two advantages. First, it assists in the designing of targeted recovery procedures which are lightweight and highly effective. Second, providing incomplete information to an attacker (by hiding detection information) prevents any sort of strategy refinement by the attacker. Because the recovery is delayed, the system relies on replication to account for the intermediate system unreliability. Replication provides reliable alternatives to suspected (but unrecovered) replicas. Though costly, redundancy and replication are necessary requirements for the fault-tolerance aspect of mission survivability. This design merely extends its use to address security.

The presented mission survivability solution is in a way, a game changer. Instead of relying on a generic mix of traditional solutions, it assists in the designing of a more effective (targeted) recovery in response to an attack. It buys the system time to figure out AIOS while the attacker remains ignorant about any detections. Extensive use of heavyweight solutions such as honeypots is successfully avoided in this design. Note that no current or prior survivability solutions work in this manner. The general rule is to detect anomalies or attacks and raise alerts that initiate recovery. However, in this solution, the reaction is delayed (can afford it in long running missions) and losses are covered via replication so more time can be bought to profile an attack and earn all the aforementioned advantages.

To summarize, this chapter extends SWAT to work effectively in a distributed environment. It involves tamper-resistant and surreptitious detection and node-to-node verification of suspicious events. Distributed SWAT aims to identify AIOS and design targeted recoveries that promote survivability. Its security strength has been theoretically analyzed while the performance and scalability aspects are measured via JavaSim simulations. Simulations demonstrate distributed SWAT's high scalability with respect to network size and application runtime. It is further realized that the time overhead for long running applications can be easily kept under 1% of original runtime by carefully adjusting the security strength.

Major contributions of this chapter are:

- Extending SWAT for its efficient and effective adoption in a distributed environment.
- Analyzing distributed SWAT for its security and tamper-resistance properties.
- Evaluating the performance of distributed SWAT.

Section 5.2 presents some preliminaries. Section 5.3 describes the architecture and other details of distributed SWAT. Evaluation of the framework is described in Section 5.4, followed by a discussion and conclusion in Section 5.5.

5.2 Preliminaries

This chapter builds upon the mission survivability research from Chapter 3 wherein the solution approach relied upon a centralized and replicated architecture. In this centralized version of SWAT, all replicas are assumed to be running in lockstep and are regulated by a central authority, also called the *coordinator*. The coordinator distributes workload among these replicas, each running on separate hardware (or node). A host-based intrusion detection system (IDS) such as Tripwire [62] monitors each node for signs of intrusion. Any intrusion attempts are translated to an *integrity-signature* and stored away in the hardware where it stays secure and hidden from the attacker. To make this framework lightweight, modified test-logic such as design for testability (DFT) or built-in self-test (BIST) is used. This pre-existing hardware comes virtually at no extra cost. Replicas send periodic checkpoints/intermediate results to the coordinator for verification. The coordinator responds back with a *go-ahead* signal if nothing out-of-ordinary is detected. The integrity-signatures stored in the hardware piggyback these checkpoints and are surreptitiously submitted to the coordinator for verification. Thus, coordinator knows about the possibly compromised replicas and can easily ignore their submissions. However, it still sends a go-ahead signal to all the replicas. The idea is to hide any detection information from an attacker until the extent of a compromise is fully known to the administrator. This prevents any unfavorable reactions by an aggressive or desperate attacker (as discussed previously). Under APT, attackers usually have multiple footholds inside the system. Administrators need a complete picture if they intend to fully recover a system and should avoid generic solutions such as a secure reboot. Rebooting can easily disrupt a mission's continuity and timeline. Frequent rebooting can even lead to denial-of-service (DoS) attacks. Additionally, if a system is restored to the same state that was earlier compromised, there are no guarantees that it will not be compromised again.

The research in Chapter 3, referred to as the *base scheme* in the sequel, provides a centralized framework for SWAT. In order to extend centralized SWAT for its effective adoption in a distributed environment, the following three goals can be defined:

- To make security-monitors at each node tamper-resistant? – Extension 1
- To ensure secure communication between nodes? – Extension 2
- To extend centralized SWAT to a distributed environment? – Extension 3

Extension 1. IDSes are usually deployed in user-space making them as vulnerable as the processes they monitor. In Chapter 4, a coveillance-based watchdog solution is presented for this classic problem of 'Who watches the watcher.' This solution is chosen over other virtual-machine based solutions because it is lightweight and effective, thus conforming to mission survivability requirements. Moreover, it performs equally well on both uni-core and multi-core platforms. This scheme involved lightweight processes called *process-monitors* monitoring each other in a cyclic fashion. Since cycles do not have any loose ends, all participating process-monitors are monitored at all times. Refer to Fig. 4.4 for a diagram of the ring topology. In a uni-core environment, all process-monitors in the ring run on the same processor. A process-monitor from this ring is assigned an additional responsibility of monitoring the security monitor (e.g., Tripwire). Any tampering of the security monitor (or process-monitor) is reported by the process-monitor monitoring it and is captured as an integrity-signature stored in the hardware (as discussed in the base scheme). There are several other advanced topologies described in Chapter 4. However, for the sake of simplicity, this chapter employs ring topology.

Extension 2. As described in the base scheme, test-logic circuitry is used to store integrity-signatures at the hardware level. Because test-logic does not provide any processing capabilities at all, additional software is needed for secure communication if test-logic circuitry is used for storage. This new software can be made tamper-resistant by using the coveillance-based watchdog framework (Extension 1). Any cryptographic keys used by this software should be stored at the hardware level. If the software or its process-monitors are attacked, instead of updating the integrity-signature in the hardware, all cryptographic keys can be deleted, thereby disabling any impersonation by the attacker. Alternatively, TPM can be used. TPM is a secured micro-controller with cryptographic functionalities. It enables the execution of cryptographic functions within the hardware. Its capabilities include random number generation, RSA encryption/decryption, SHA-1 hash calculations and limited NVRAM [11]. TPM contains 16 internal memory slots called platform configuration registers (PCRs) that are initialized to known values. A function call PCRExtend is the only way a software can update PCR values. Once updated, these values are securely stored in the TPM. Thus, PCRExtend can be used by the security monitor to commit an integrity-signature for storage in the hardware. API calls like Seal/Unseal/Bind/Unbind/Quote, etc. can be used to avail authentication, confidentiality and integrity features provided by the TPM [134].

Extension 3. For this extension, a novel solution is presented in Section 5.3.

5.3 Distributed Architecture for Survivability

5.3.1 Framework Requirements

Following are the requirements for a good distributed survivability solution:

- *Should be highly adoptable in a distributed environment.* With the advent of cloud technology, any survivability solution cannot be considered for generic use until it runs efficiently in a distributed environment.
- *Should support the commercial off-the-shelf (COTS) paradigm.* A good solution should also not restrict its use to a closed system or have security policies that restrict the set of systems it can communicate with.
- *Should be tamper-resistant.*
- *Should deal with the APT-based attack model described in Chapter 3.* As a result, it should consider the repercussions before acting in a certain way or divulging new information.
- *Must fulfill all mission survivability requirements.* It should preserve mission's timeliness property. However, predictable and bounded delays are allowed and can be accounted for in the original mission timeline.

5.3.2 Framework Details

Distributed SWAT leverages concepts of deception, especially information hiding [146, 28]. As discussed in Section 5.1, it is important that any intrusions are reported surreptitiously. The monitoring entity can then decide whether to take immediate action or to continue observing in order to refine the attack profile.

Good attack profiles help in the design of targeted recovery procedures that are lightweight and effective. For instance, if the actual source of a malicious event is discovered, a system reboot (which can disrupt mission's continuity and timeline) may no longer be necessary. To achieve these objectives, a system needs to have the following characteristics:

- *Tamper-resistant monitoring of critical processes at each node:* For this purpose, Extension 1 as described in Section 5.2 can be used. All critical processes (such as security monitors) in the user-space are provided with tamper-resistant monitoring using a lightweight, cyclic watchdog framework.
- *Tamper-resistant and surreptitious capture and storage of integrity-signature at each host:* This can be achieved via the use of either TPM or test-logic circuitry as discussed in Extension 2 of Section 5.2.
- *Tamper-resistant and surreptitious intrusion reporting to a remote authority:* This can be achieved using TPM or a combination of software and test-logic circuitry as discussed in Extension 2 of Section 5.2.

In order to possess all these characteristics, each node in the distributed SWAT needs to have the following components as shown in Fig. 5.1:

- *Hardware Replicas:* Replication provides the redundancy required for fault-tolerance. Redundancy is the cornerstone of survivable systems. SWAT delays recovery (or any other response) until it has assessed the extent of system compromise. Replication can help a system survive this delay in response. Furthermore, even if some replicas are compromised near mission completion, mission can still survive with the help of tamper-free replicas.

- *Effective isolation of replicas:* It is essential that replicas are diverse and well-isolated to increase their robustness against repeated attacks. An ideal environment should have each replica running on a separate physical machine. However, virtual machines are good alternatives as well. Although replication increases the overhead but it is a necessary requirement for mission survivability.

  If virtual machines (VM) sharing the same hardware are used as replicas, they would need to share a single TPM or test-logic circuitry for intrusion reporting. This sharing can be implemented by assigning dedicated bits to indicate the integrity-signature of each replica. For instance, a simple implementation can use a bit marked '1' to indicate tampered state and '0' to indicate tamper-free state of a replica. This way, a 32-bit word can track 32 virtual replicas running on a single machine. So, a word 0000.....0101 denotes that replicas 1 and 3 may have been tampered with. Considering that PCRs in TPM have a capacity of 160 bits each, they can store richer information than just a binary security status. However, this is an application-specific design issue and is outside the scope of this dissertation.

- *Hardware-support for secure and surreptitious storage of integrity-signatures:* For the sake of simplicity, TPM is used in this design. Note that any process that writes to the TPM should be verified for integrity. Secure boot and trust chains facilitated by TPM can easily solve this problem [134].

- *Tamper-resistant monitoring for all critical processes in the user-space:* For this purpose, the cyclic watchdog framework described in Extension 1 of Section 5.2 is employed.

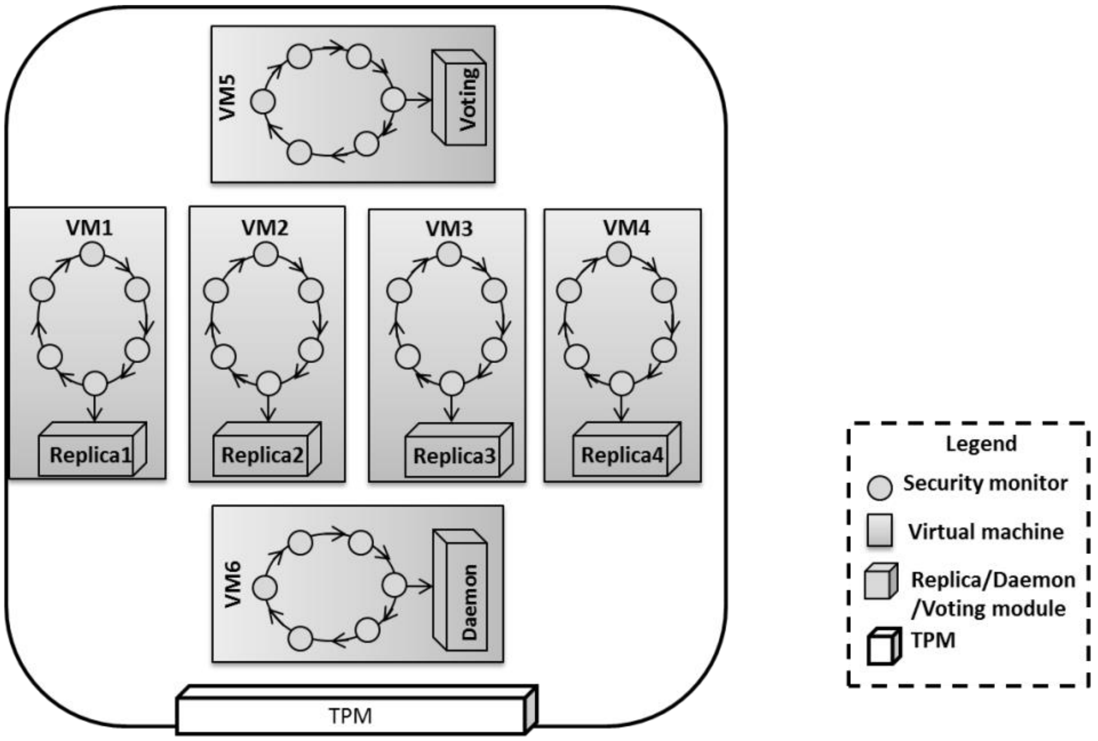

Figure 5.1: Various sub-components of a node in the proposed distributed survivability solution

- *TPM daemon:* TPM does not know what to do with the data that it stores and processes. A TPM daemon is the logical layer over TPM that provides useful functionalities. For instance, parsing and structuring the data decrypted by TPM to a form usable by other applications, scheduling tasks to TPM, etc.

- *Voting service:* Based on the information provided by the TPM daemon, the voting service decides which replicas to trust. From among the results reported by these trusted replicas, a voting (majority, random selection, etc.) is conducted to determine the final result.

5.3.3 Operational Details

Let S be a distributed network of nodes $\{S_i\}_{i=1}^{n}$ running a particular mission critical application such as a process control system or a peer-to-peer computation requiring a certain level of mission assurance. Each node S_i calculates an intermediate result and submits it to another node in the network. Every node has m replicas running the same application in lockstep. Therefore, it submits $\{\rho_j\}_{j=1}^{m}$ copies of a result. A node can run in any one of the following modes: Process, Send, Receive and Vote. The operational steps are formally described in Fig. 5.3.

In 'Process' mode, a few threads (δ) run replicas while others run a tamper-resistant cyclic watchdog framework monitoring each replica. A participating node combines input received from another node ($\hat{\rho}$), if applicable, with its own data. This data is then processed by the local replicas. Meanwhile, if the monitoring framework senses intrusion, it commits a change in the value of integrity-signature ψ stored at the hardware level. After the processing at all replicas is complete, the node switches to 'Send' mode. In this mode, it securely sends the set of results $\{\rho_j\}_{j=1}^{m}$ along with the integrity-signature ψ to the next node as determined by the application. Note that this secure communication between nodes is TPM-assisted. If the node has any further tasks pending, it switches to 'Receive' mode and waits for further input; otherwise, it stops processing.

Upon receiving data, a node switches to 'Voting' mode. Here, it uses ψ to derive a set C of replicas suspected to be compromised at the sending node. It disregards all votes from these replicas and performs voting on the rest to derive the result $\hat{\rho}$. It then switches to 'Process' mode. At startup, all nodes are either in the 'Receive' or 'Process' mode depending on the design.

An instance of such a mission survivable architecture is shown Fig. 5.3. This

```
while (System = ‘Running’) do
    if (MODE = ‘Process’) then
        Run parallel threads $\delta$ for replicas and process-monitors
        if ($\delta$ = ‘Replica’) then
            if (Pending tasks $\neq$ NULL) then
                if ($\hat{\rho} \neq$ NULL) then
                    Combine $\hat{\rho}$ with node’s data
                end if
                Feed data to replicas for processing
                Wait for processing to complete
            end if
            Switch to MODE = ‘Send’
        end if
        if ($\delta$ = ‘Process-Monitor’) then
            while (TRUE) do
                if (Suspicious activity detected) then
                    Change the value of $\psi$ accordingly
                end if
            end while
        end if
    else if MODE = ‘Send’ then
        SecureSend($\{\rho_j\}_{j=1}^{m}$), $\psi$)
        if Pending tasks $\neq$ NULL then
            Switch to MODE = ‘Receive’
        else
            END
        end if
    else if MODE = ‘Receive’ then
        Receive($\{\rho_j\}_{j=1}^{m}$, $\psi$)
        Switch to MODE = ‘Vote’
    else if MODE = ‘Vote’ then
        Analyze $\psi$ to identify C, a set of compromised replicas on the sending
        node
        Vote on $\{\rho_j\}_{j=1}^{m}$ where j $\notin$ C
        Switch to MODE = ‘Process’
    else
        Error: Unsupported MODE
    end if
end while
```

Figure 5.2: Pseudocode - Operational details

figure depicts a sample peer-to-peer application consisting of four geographically distributed nodes which represent processing elements (PEs) and a client (laptop). All PEs gather data from their respective environments. They process this data and forward a set of results to their peers chosen by the application. The receiving PE combines the received information with its own data, processes it and forwards its set of results to the next PE. Node at location 1 has several replicas running the application. These replicas are being monitored by their respective cyclic watchdog frameworks that commit an integrity-signature to the TPM if suspicious activity is detected. A set of results from Node 1 is sent to nodes at locations 2 and 3. Along with the set of results, the daemon sends over TPM information as well. The TPM daemon at the destination node parses this information and delivers it to the voting service. Voting service determines which results (replicas) are to be trusted and uses them in further processing. Such exchanges go on until a set of results is finally delivered by node 4 to the client. Client, in this case, needs to run a TPM, a daemon and a voting service in order to get the final result.

5.4 Evaluation

Distributed SWAT is evaluated for its security strength and performance. Section 5.4.1 models security relationships between its various components and assists in analyzing SWAT's security strength. Section 5.4.2 describes SimJava simulations used to evaluate SWAT's time performance.

5.4.1 System Modeling

In this section, the security dependencies are modeled for analyzing the security strength of distributed SWAT using propositional logic as described by Schryen et

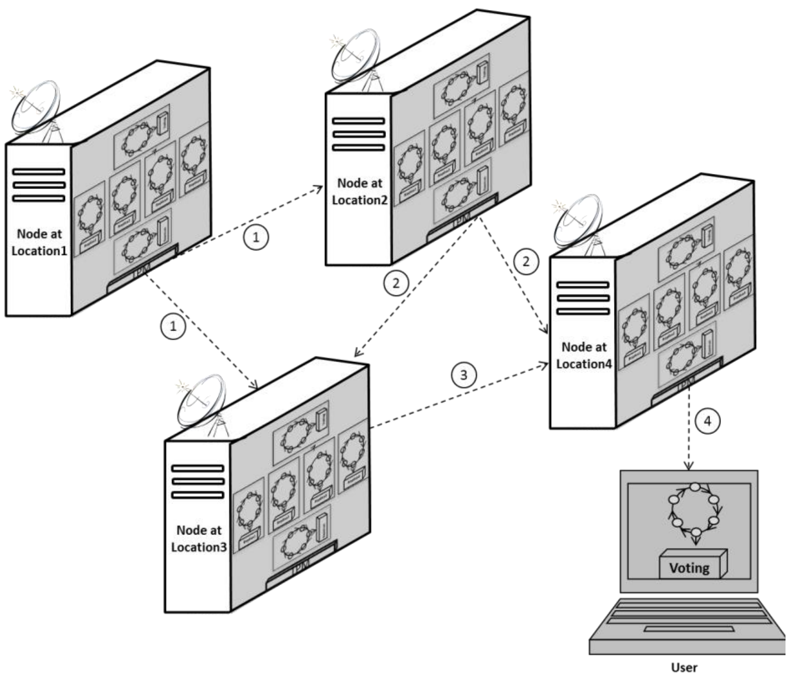

Figure 5.3: A distributed architecture for mission survivability against APT

al. [121]. The end goal is to determine the trustworthiness of a distributed system that employs SWAT to deliver correct results while fulfilling a set r of certain security requirements.

Each node S_i, where $1 \leq i \leq n$, has the following components (as described earlier):

- m number of replicas running the same application on separate physical or virtual machines, collectively represented as $\{R_j\}_{j=1}^{m}$. Each replica is monitored by a cyclic watchdog framework $\{P_k\}_{k=1}^{\{c_l\}_{l=1}^{m}}$, where P_k is a process-monitor and c_l is the total number of process-monitors monitoring each replica. This setup is easily adoptable by both uni-core and multi-core platforms.

- TPM hardware at node i, represented as T_i.

- TPM logical layer daemon at node i, represented as D_i, monitored by a cyclic watchdog framework $\{P_s\}_{s=1}^{a}$, where P_s is a process-monitor and a represents the number of participating process-monitors.

- Voter at node i, represented as V_i, performs voting functions and is monitored by a cyclic watchdog framework $\{P_t\}_{t=1}^{b}$, where P_t is a process-monitor and b represents the number of participating process-monitors.

When a client submits a request to this distributed system, it trusts that all its nodes will satisfy r.

$$S = S_1 \wedge S_2 \wedge S_3 \wedge ...... \wedge S_n \tag{5.1}$$

For each S_i to fulfill r, all its components should satisfy r.

$$S_i = T_i \wedge D_i \wedge V_i \wedge R_1 \wedge R_2 \wedge R_3 \wedge ...... \wedge R_m \tag{5.2}$$

Tamper-resistant functioning of these components relies on the tamper-resistant functioning of their respective coveillance-based watchdog frameworks. Thus,

$$\begin{aligned} S_i = T_i \wedge \underbrace{\{P_1 \vee P_2 \vee ...... \vee P_a\}}_{D_i} \wedge \underbrace{\{P_1 \vee P_2 \vee ...... \vee P_b\}}_{V_i} \wedge \\ \underbrace{\{P_1 \vee P_2 \vee ...... \vee P_{c_1}\}}_{R_{i_1}} \wedge \underbrace{\{P_1 \vee P_2 \vee ...... \vee P_{c_2}\}}_{R_{i_2}} \wedge ...... \wedge \\ \underbrace{\{P_1 \vee P_2 \vee ...... \vee P_{c_m}\}}_{R_{i_m}} \end{aligned} \tag{5.3}$$

should satisfy r.

In a cyclic watchdog configuration, there is a high probability that all process-monitors are monitored at all times (discussed in Chapter 4). Thus, they cannot be tampered without raising an alert. Therefore, they can be assumed to be tamper-resistant with a probability approaching 1.

$$S_i = T_i \wedge \underbrace{\{1\}}_{D_i} \wedge \underbrace{\{1\}}_{R_{i_1}} \wedge \underbrace{\{1\}}_{R_{i_2}} \wedge \underbrace{\{1\}}_{R_{i_3}} \tag{5.4}$$

Combining (5.1), (5.2) and (5.4),

$$S = \{T_1 \wedge T_2 \wedge T_3 \wedge .... \wedge T_i\} \tag{5.5}$$

should satisfy r.

From (5.5), it can be concluded that the reliability of distributed SWAT depends on the reliability of its TPM components to satisfy r. In general, TPMs are considered to be highly reliable [11] which indicates the high reliability of distributed SWAT.

5.4.2 Simulation

A generic peer-to-peer (P2P) distributed network that employs distributed SWAT is simulated for performance evaluation. The simulated network topology is shown in Fig. 5.4. All nodes are connected to one another directly or through a string of peers. Clients can submit their tasks to any of the participating nodes and get a result back from the same node.

The performance evaluation is conducted using SimJava (2.0) on a 32-bit Windows system with 2 GB RAM and Intel Core Duo processor. Simulation parameters are derived using the multi-step evaluation approach as previously seen in Chapter 3. These simulations primarily evaluate time performance and scalabil-

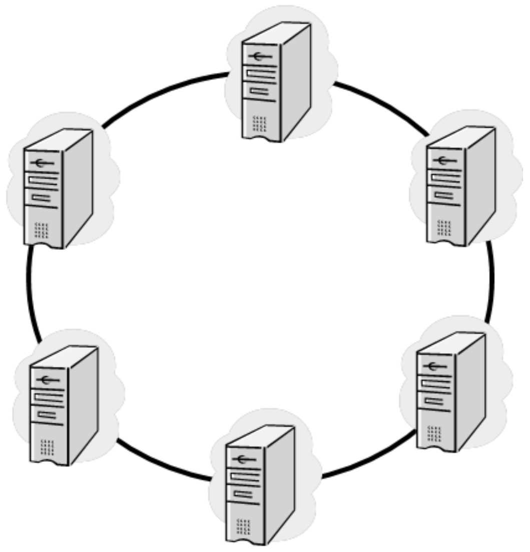

Figure 5.4: Peer-to-Peer topology of the simulated distributed network

ity. Space overhead is not particularly evaluated because the memory requirement is not significant if replicas are already a part of system's fault-tolerance setup. Furthermore, all the other system components (daemons and TPM) are extremely lightweight in terms of memory.

Long running missions are primarily considered for evaluation because APTs usually have a greater impact on them. Such missions, if interrupted at a later stage, require lots of resources and time for re-runs (if re-runs are possible at all). Short-missions can always be reset, recovered or replayed if found corrupted or compromised.

Figure 5.5 depicts the scalability of distributed SWAT with respect to mission's runtime. Thus, the x-axis represents average runtime per node while the y-axis represents time overhead as a percentage of original mission runtime on a logarithmic scale. This simulation is replayed for five different network sizes. For evaluation purposes, simple majority voting is assumed. In Fig. 5.5, a simple challenge-response mechanism (instead of RSA) is used for authenticating (but

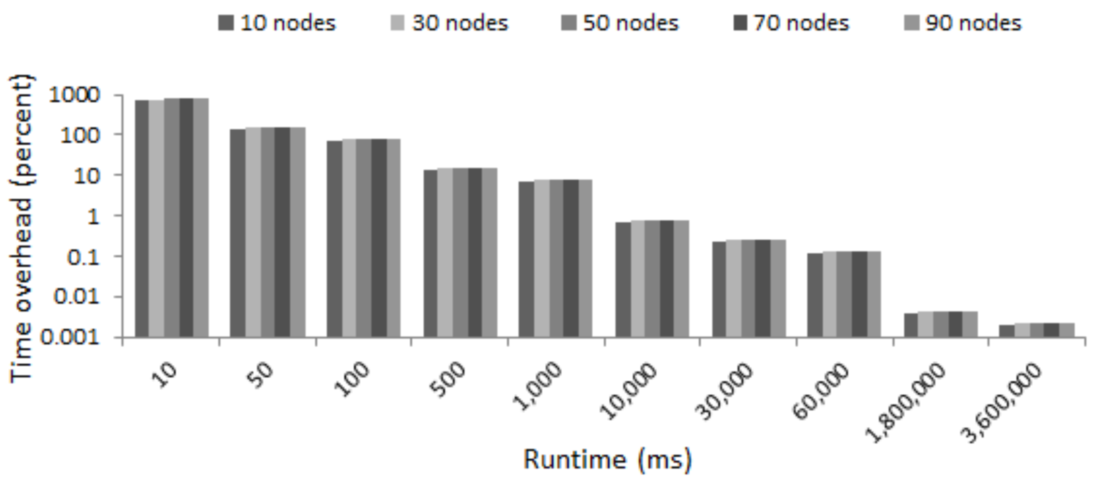

Figure 5.5: Time overhead for different network sizes in the absence of RSA-based TPM security

not encrypting) the communication. A problem with this communication security setup is that it assumes a weak APT model where the attacker has less or no control over the communication channel. It can be observed that the (time) overhead percentage decreases with increasing mission runtime. This is because for a fixed network size, time overhead increases at a much slower rate than mission runtime, making SWAT extremely scalable. However, as the network size increases (from 10 to 90 nodes), a slight increase in time overhead can be noticed. This scalability aspect is investigated shortly. A major takeaway from this simulation is that for any network size, missions with runtime greater than 1 sec (average per node) incur a time overhead of less than 10%. Furthermore, missions with an average runtime of greater than 10 sec per node have a time overhead of less than 1%. This number decreases further as the mission runtime increases.

Employing RSA for communication security (using TPM) incurs additional time overhead. Measurements received after encrypting the communication in previous simulation with an RSA key-size of 512 bits and data-size of 20 bytes are shown in Fig. 5.6. Here, time overhead observed follows the same trend as in Fig.

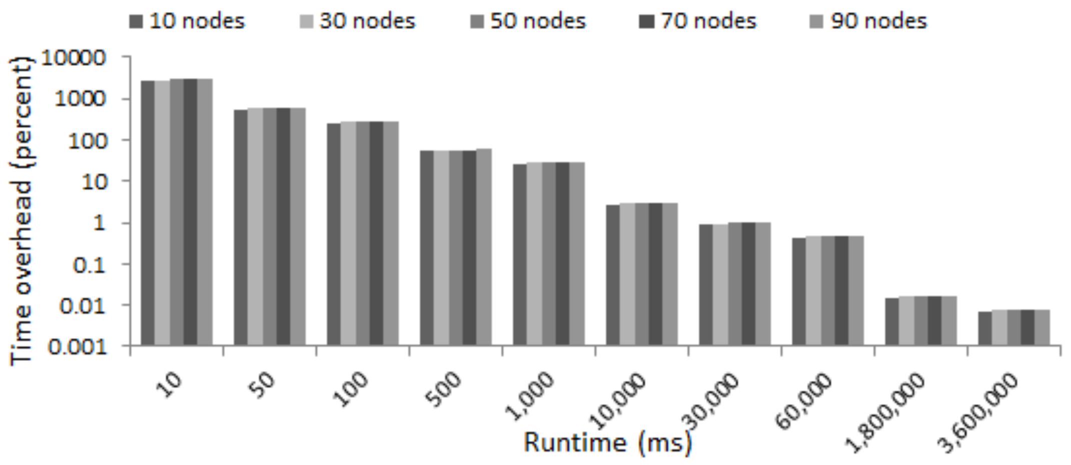

Figure 5.6: Time overhead for different network sizes in the presence of RSA-based TPM security

5.5. However, the 10% time overhead cutoff is for missions where average runtime per node is 10 sec or greater. The 1% time overhead cutoff is for the missions with 30 sec or greater of average runtime per node.

Figure 5.7 analyzes SWAT's scalability with respect to the network size (number of nodes). Thus, the x-axis represents network size while the y-axis represents time overhead as a percentage of original mission runtime on a logarithmic scale. This simulation uses a simple challenge-response mechanism for authentication (not RSA). It is observed that the time overhead increases drastically as the network size expands from 3 to 10 nodes; it increases very slowly afterwards. Note that the time unit here represents average runtime per node. This means that when a bigger network is considered, it results in a longer runtime. This explains the trend seen for this simulation. For a mission with an average runtime of 1 sec or greater per node, the time overhead is bounded by 10% irrespective of the network size. For a mission where the average runtime per node is 10 sec or greater, this bound decreases to 1%. These observations demonstrate the high scalability of SWAT for long running missions irrespective of the network size.

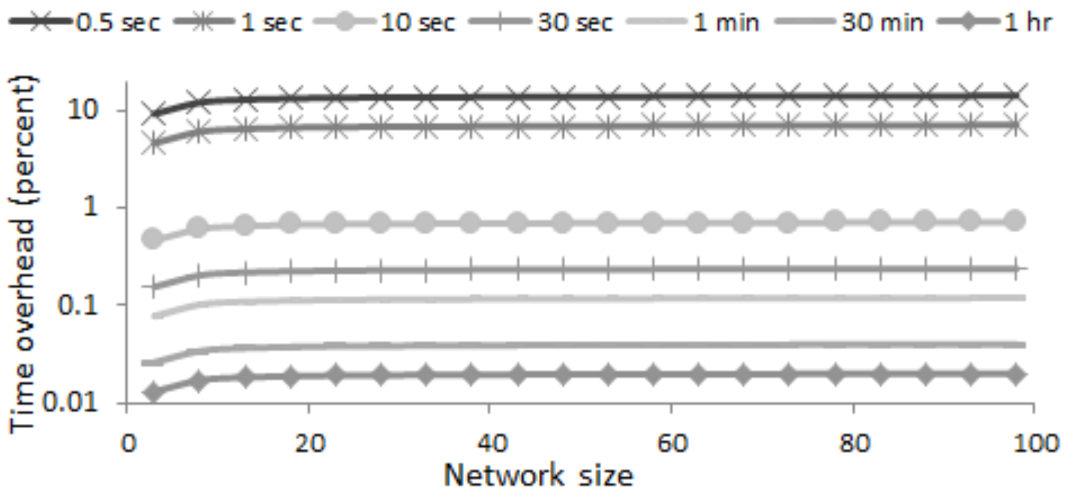

Figure 5.7: Time overhead for different average mission runtimes (at each node) in the absence of RSA-based TPM security

Figure 5.8 measures the same parameters as measured in Fig. 5.7, except that it uses RSA for communication security. It uses a key-size of 512 bits and data-size of 20 bytes. Same trends are observed in both cases as shown in Fig. 5.8 and Fig. 5.7. However, because of the increased time overhead, for achieving the 1% bound, a mission needs to have at least 30 sec of average runtime per node.

Note that a specific key and data size are used for the evaluations above. If TPM is required to provide better security strength, it may need to use stronger RSA parameters. Figure 5.9 measures the time overhead for the different levels of RSA-based TPM security. The x-axis represents network size and the y-axis represents time overhead as a percentage of original mission runtime on a logarithmic scale. Key-size and data-size are the two parameters that determine security strength of RSA. Table 5.1 lists the various cases of RSA security settings for TPM. Case 1 refers to the scenario with challenge-response based authentication and no RSA. All the rest of the cases use RSA with the specified parameters. It can be observed that there is a slow increase in overhead when data-size increases and key-size

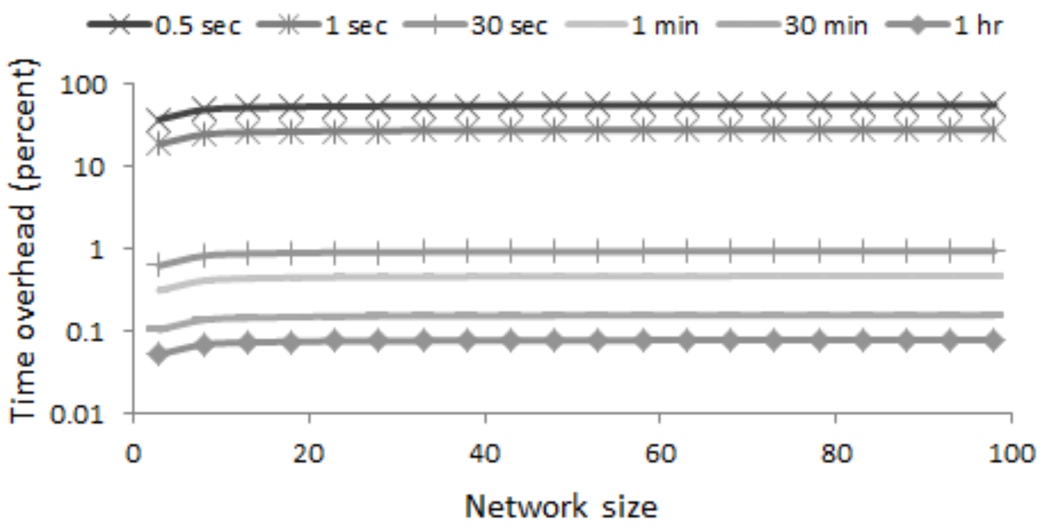

Figure 5.8: Time overhead for different average mission runtimes (at each node) in the presence of RSA-based TPM security

remains the same. However, an increase in key-size increases the time overhead drastically. Cases 2, 5 and 8 with steeper positive slopes of time overhead mark the increasing key-size. All other cases with a gradually increasing slope are for the increasing data-size. Note that the time overhead shoots up as the key-size reaches 2048, which provides very strong security but may cause excessive time overhead.

5.5 Conclusion

This chapter presents a mission survivability solution that aims to survive APTs in a distributed environment. The basic idea is to buy defense enough time to figure out the precise nature of an attack. This helps in the designing of targeted recovery procedures. Targeted recoveries are more effective and incur lesser overhead than generic procedures. The solution also helps to disguise the knowledge of detection

Table 5.1: Cases for different levels of RSA-based TPM security

	RSA key-size (bits)	Encryption data-size (bytes)
Case 1	-	-
Case 2	512	20
Case 3	512	50
Case 4	512	100
Case 5	1024	20
Case 6	1024	50
Case 7	1024	100
Case 8	2048	20
Case 9	2048	50
Case 10	2048	100

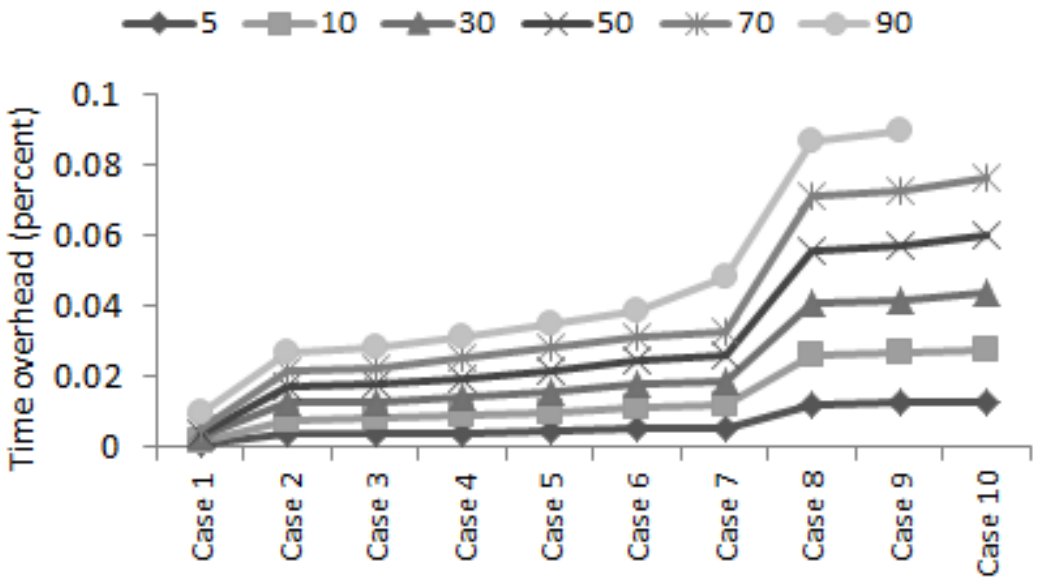

Figure 5.9: Time Overhead plotted against levels of security provided by TPM for different network sizes

from the attacker. This 'false assurance' could prevent the attacker from switching to an aggressive strategy or contingency plan.

Each node in SWAT runs application replicas in lockstep but on separate and isolated virtual machines. They also employ IDSes that surreptitiously report suspicious activities to the TPM (in the form of integrity-signatures). Nodes exchange sets of replica-results and their integrity-signatures for verification by other nodes. All the user-space components of distributed SWAT that require tamper-resistance employ the watchdog cyclic framework described in Chapter 4. Compromised replicas are not recovered immediately but are monitored for more information about AIOS. Meanwhile, the intermediate results are obtained from the tamper-free replicas (with good integrity-signatures). This buys the defense more time to design targeted recoveries without leading to a change in attacker's strategies or risking mission survivability. Note that the delay in reaction introduced by SWAT is contingent upon the threat situation. If SWAT faces an immediate threat and can no longer wait, it may even initiate a generic recovery procedure. Hence, SWAT is adaptive in nature and waits for more information only when possible without compromising the survivability of the mission.

The security strength of the presented solution is analyzed using theoretical system modeling. It is concluded that the security strength of distributed SWAT relies primarily on subsystems that are inherently secure. SimJava simulations are used to evaluate a generic P2P network that employs distributed SWAT. Measurements indicate that distributed SWAT incurs a time overhead that constitutes only a fraction of mission's runtime. This fraction decreases drastically as the mission runtime increases. For a mission that runs for at least 1 sec average per node in the network, a time overhead bound of 1% can be easily achieved by adjusting its security strength. Thus, long running missions can efficiently adopt SWAT without the risk of over-stretching their timelines. Because the response to an attack is delayed, a maximum bound on the expected number of recoveries can be proactively calculated and accounted for in the original mission timeline. This prevents the reactive recoveries, whenever initiated, from violating mission survivability's timeliness property. Strong security and tamper-resistance properties of distributed SWAT, along with its efficiency, could make it a good defense strategy against APTs for which no good solutions currently exist.

In the next chapter, we discuss the efficient deployment of deception-based solutions in a production environment and how deception can assist in effective identification of zero-day exploits.

Chapter 6

SWAT in Production Environment

6.1 Introduction

The preceding chapters in this dissertation have presented centralized and distributed versions of SWAT (Survivable framework for (cyber) Warfare against Advanced Threats) for strengthening mission survivability against advanced persistent threats (APT). Considering that APTs specifically target critical systems, they tend to exploit zero-day vulnerabilities often. Usually employed anomaly-based detection solutions have a high overhead and thus, are not suitable for production environments. This chapter presents an extension to SWAT that enables it to defend against zero-day exploits by employing deception at the end systems. Deception employed in this chapter uses honeypots and is customized in real-time. In addition, it addresses the generic problem of deploying deception-based survivability solutions in a production environment. End systems in production environment have a necessary requirement to stay responsive and secured at all times while openly interacting with the outside world. Thus, they are the most vulnerable but have a great vantage point. The solution presented in this chapter leverages the location of end-systems for increased effectiveness. Unlike the deception-based solutions that pass the entire traffic through honeypots, the presented solution sieves out suspicious traffic based on its behavior. Each design choice is supported by evidence and a detailed review of related literature. Finally, the challenges involved in implementing this SWAT extension and their possible resolutions are presented.

6.2 Related Work

Several anomaly-based detection systems have been proposed in order to detect zero-day exploits [25]. However, Liu et al. [80] describe the big challenge, "how to make correct proactive (especially predictive) real-time defense decisions during an earlier stage of the attack in such a way that much less harm will be caused without consuming a lot of resources?" Solutions that attempt to recognize a zero-day, multi-shot stealth attack usually take two approaches – predictive and reactive. Under the predictive approach, all the suspected policy violations are taken as a sign of intrusion. During the early stages of an attack, false alarms are difficult to sieve out, thus this approach may result in service degradation or even resource exhaustion. Under the reactive approach, defender relies on the correlation techniques employed by the system and takes an action only when he is somewhat certain of the foul play. As with any such approach, it is difficult to know when to react. If the alarms are not considered unless a complete attack profile emerges, it may be too late to react. Thus, the defender needs to find a balance between these two extreme ways of dealing with zero-day exploits. Such a trade-off is offered by solutions based on honeypots (a form of deception). Usually, the defender redirects all the traffic through honeypots, which is responsible for white-listing the traffic [110, 105]. Many researchers have introduced methodologies for employing honeynet in a production-based environment [76, 69]. However, analyzing heavy amounts of traffic using honeypot-like systems requires a lot of resources and time.

Anagnostakis et al. [4] proposed shadow honeypots as an effective solution to deploy honeypots in a production environment. Shadow honeypot is an identical copy of the production server but instrumented to detect potential attacks. A variety of anomaly detectors monitor traffic in the network and the suspected traffic is forwarded to the shadow honeypot. Misclassified traffic is verified by the shadow and transparently handled correctly. There are many challenges with this approach. First, predictive anomaly detectors (higher sensitivity) will have more false positives and will direct more misclassified traffic to the shadow honeypot, creating unnecessary delays and overhead. Reactive anomaly detectors (lower sensitivity) will take more time to create a complete profile and will miss a lot of malicious traffic before identifying a problem with the flow. Moreover, identifying zero-day attacks ask for higher sensitivity intrusion detection. Additionally, each suspected traffic stream may need separate copies of shadow honeypot (else an attacker can verify deception by initiating two parallel malicious flows). This further increases the overhead.

6.3 SWAT Extension

6.3.1 Extension Modeling

Honeypots are deceptive systems that come across as systems capable of low-resource compromise with high perceived gains. Honeypots not only distract attackers from attacking the main system but also log any activities extensively. Studying these logs can help a defender gauge AIOS (attacker intent, objectives and strategies) and design good defense strategies to ward off any future attacks. Spitzner [128] describes honeypot as “a security deception resource whose value lies in being probed, attacked, or compromised.” Honeypots are generally classified under two categories – *physical* and *virtual*. Physical honeypots are created using real computer systems while virtual honeypots use software to emulate the workings of a real honeypot and the connecting network. Virtual honeypots are cheaper to create and maintain and hence, are used in production environments more often. Virtual honeypots are further divided into high interactive and low interactive honeypots. Qasswawi et al. [113] provide a good overview of the deception techniques used in virtual honeypots.

High interactive honeypots emulate real operating systems. Thus, attackers feel like they are interacting with real systems and can even completely compromise them. Some examples are User Mode Linux (UML), VMware and Argos. Low-interaction honeypots simulate limited network services and vulnerabilities. They cannot be completely exploited. Examples are LaBrea, Honeyd and Nepenthes [112, 113]. Cohen’s Deception Toolkit (DTK) [27] laid the groundwork for low-interaction honeypots. It led to the development of advanced products such as Honeyd. Honeyd [128] simulates services at TCP/IP level in order to deceive tools like Nmap and Xprobe. Though it does not emulate an entire operating system, its observables are modified to give the impression that it does.

Honeypot farm is a cluster comprised of honeypots of the same or diverse kinds. Hybrid honeypot farms are usually a mixture of low and high interactive honeypots.

Designing deception-based prevention is the first step in the SWAT extension modeling. Traditional preventive measures include firewalls, encryption techniques, digital certificates, strong passwords, access control, strong configuration management and training programs. These measures are known to be effective in deterring weak adversaries. However, most advanced adversaries manage to find a way around these measures. McGill [88] suggested an interesting approach towards attack prevention. Instead, of treating attacks as constant probabilistic events such as earthquakes or tsunami that cannot be prevented, they should be treated as 'conditional risk.' This is a good representation of real-world attacks because attackers usually take calculated risks. They start with snooping around the system looking for weaknesses and vulnerabilities to exploit. Thereafter, they measure the resource requirement of the possible attacks against their available resources. Based on similar analyses, attackers decide whether to go ahead or abandon the attack. Thus, the apparent vulnerability of a system (in other words, the resource requirement of attacking the system) heavily impacts the probability of it being attacked. Thus, deceiving an attacker into believing that a system has stronger security is one of the many deceptive prevention measures. Based on the vast amount of literature on the factors influencing an attacker's choice of target, deception-based prevention methodologies can be categorized under the following three headings:

- *Hiding:* Hiding is one of the simplest forms of deception. One could use schemes such as fingerprint and protocol scrubbing to hide important system information [141, 125]. In a similar manner, false information can be deliberately fed to the adversaries as well. Yuill et al. [146] developed a model for understanding, comparing, and developing methods of deceptive hiding.

- *Distraction:* McGill [88] demonstrated that given two targets of equal value, an attacker is more likely to attack the target with lesser protection. However, Sandler and Harvey [120] analytically proved that this tendency continues until a threshold. If more vulnerability is introduced, an attacker's preference of the chosen target does not increase beyond a certain threshold. In other words, decreased protection does not always translate to an increased attacker interest. Even if an attacker is deflected from the main target by providing it with stronger security or adding a distraction target with lesser security, the attack probability on the main target will never be zero. A strong security model always assumes that the attacker is capable of knowing a system via reconnaissance, social engineering, etc. In general, attackers rely on observables to draw inferences about the system. These observables can be manipulated to feed misinformation or hide information from the attackers. Thus, strategies can be devised to affect an attacker's perception of the system and reasonable assumptions can be made about attacker's beliefs, observables and sensitivities. Studies like the one by McGill [89], model threat scenarios based on target's susceptibility and attacker's tendencies. Such models can be used to assess the attractiveness of a target to an attacker when its apparent susceptibility is manipulated.

- *Dissuasion:* Dissuasion describes the steps taken by a defender to influence adversary behavior in system's favor. It involves manipulating system observables to make it look like it has stronger security than it actually does. As discussed previously, this usually discourages attackers. Dissuasion is implemented either as deterrence or devaluation. Deterrence involves a false display of greater strength. This is generally achieved by displaying an interface with multiple intrusion detection systems or firewalls. Devaluation, on the other hand, involves manipulating observables to lessen the perceived value that may come out of a system compromise. McGill [88] developed a probabilistic framework around the use of defensive dissuasion as a defensive measure.

Axiom 1: Adding more vulnerability to one of the two equal-value systems increases the likeliness (until a threshold) of an attack on the system with more vulnerability.
Axiom 2: False display of strength dissuades an attacker from attacking the system.
Axiom 3: Increasing or decreasing the perceived value of a system affects the attacker's preference of attacking the system favorably or adversely, respectively.

Note that deception-based prevention techniques are complementary to conventional prevention techniques and not a replacement.
Designing deception-based detection is the next step in the SWAT extension modeling. If an attacker gets distracted by a vulnerable system or a traffic stream, it is flagged as suspicious by an intrusion detection system (IDS) and directed to a honeypot farm. Diverse deceptions are designed for the honeypots in the farm so that a pattern is discernible even for stealthy intrusions. Attackers use observables

to infer about the vulnerabilities of a system. Stealthy attackers prefer vulnerabilities that can be quietly exploited over the ones that make any changes to the user interface. By offering them such vulnerabilities, honeypots can collect more data to generate better intrusion profiles. There are two significant challenges in developing such a solution. First, such deceptions should force or manipulate a stealthy attacker into leaving discernible and traceable patterns. Second, such a detection mechanism should be hidden lest the attacker should get spooked and execute a contingency plan (or switch strategies) for which the defender is not likely to be prepared. In addition to that, the system should be provided with basic prevention, detection and recovery abilities while conserving the timeliness property of the mission.

Thus, in general terms, deception-based detection can be designed as follows – For a given system state $s_1(t)$, there is a set ϕ_1 of suspicious actions (for instance, a possible exploit attempt). A user that chooses an action from this set is malicious with a probability p. However, he could be benign with a probability 1-p. Let system states $s_1(t)$, $s_2(t)$,....,$s_n(t)$ (where, n is the total number of system configurations) have ϕ_1, ϕ_2,....,ϕ_n as their respective sets of suspicious actions. For some system states, this set of actions can be more clearly categorized as malicious with higher probabilities p_i where, $1 \leq i \leq n$. Choosing such states more frequently helps the defender to come up with a clear user profile in a shorter time. In honeypots, a defender can choose states with higher p_i's, which means that if an attacker keeps choosing the actions from the set ϕ, his probability of being malicious ($p_1.p_2.p_3....p_n$) will cross the threshold in a shorter time. Thus, choosing and controlling these states is crucial in determining if an attacker is malicious with a higher probability in a shorter time. Thus, deception-based detection expedites the understanding of AIOS.

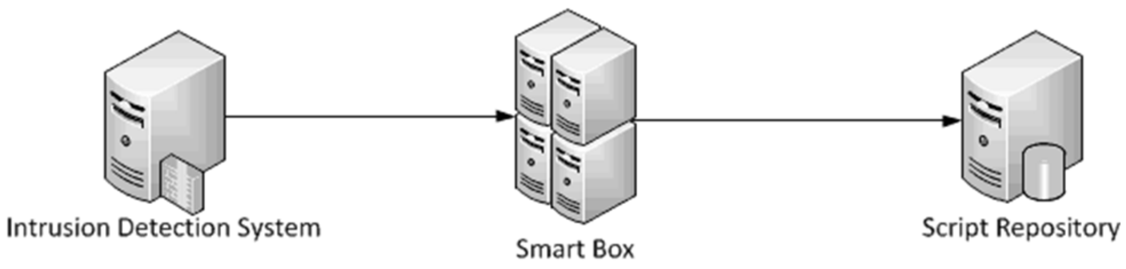

Figure 6.1: Smart-Box

The next step in SWAT modeling is designing deception-based recovery with adaptation. As discussed in Chapter 5, intrusion detection and reporting must be surreptitious and tamper-free. Any recovery should be initiated after AIOS has been estimated.

Smart-box assists in the designing of good deception strategies in real-time to generate a good AIOS profile. Conceptually, a smart-box works as shown in Fig. 6.1. It takes input from the IDS about the suspected traffic flow and generates an intermediate AIOS profile based on this information [80]. Then it maps this intermediate AIOS profile to deception scripts that can help further refine the AIOS. These scripts are stored in the script repository.

6.3.2 Design Details

Building up from the concepts discussed above, the solution architecture is presented in Fig. 6.2. It is an extension of the work presented by Lakhani [69].

As shown in Fig. 6.2, the mission survivable system runs behind several layers of protection such as firewalls, deception, etc. The first layer of proxy servers uses Axioms 1, 2 and 3 to mislead attackers into choosing systems that re-route their traffic to a honeypot farm via the smart-box. Rest of the user traffic goes through the main server, the firewall and the IDS. IDS is another layer of defense which

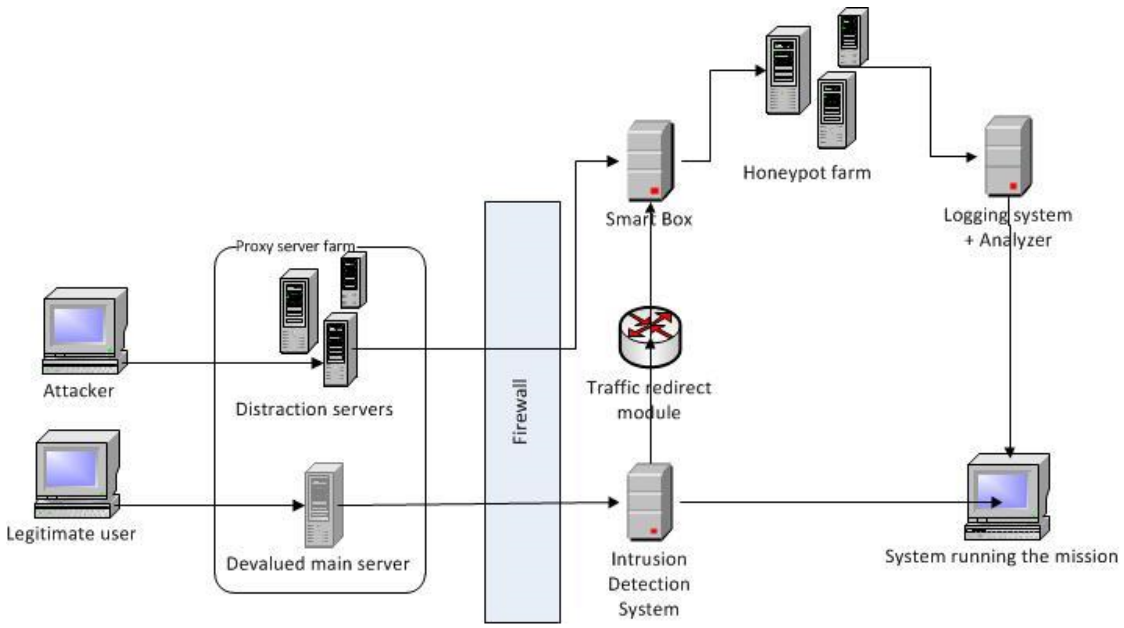

Figure 6.2: Deception framework for mission survivability

is capable of re-routing any suspicious traffic to the honeypot farm for further analysis.

In traditional solutions, a traffic stream is tagged suspicious when an IDS identifies an attack pattern associated with it or it has originated to/from the dark address space. Dark address space is the set of Internet's routable address reserved for future network expansion. These two filtration criteria worked just fine until cloud computing came along. Now attackers launch their attacks from behind the cloud using valid IP addresses and evade detection. Therefore, in addition to the two criteria, another layer of distraction proxy servers is introduced. This layer contains a main server which is widely publicized to legitimate clients. This main server is extremely secure and its apparent security is further enhanced (deception/deterrence). Thus, amateur attackers are dissuaded from attacking it. Other proxy servers expose a specific set of non-essential, vulnerable services. For instance, a server can keep the *ssh* port open to accept the traffic while another server can run a vulnerable version of Windows operating system. These vulnerable servers not only distract malicious traffic away from the main server but also provide information (about compromise attempts) to the smart-box that aids in the development of AIOS.

Another important use of smart-box is its ability to optimize resource allocation in hybrid honeypot farms. Honeypots should be assigned to traffic streams based on their immediate or predicted AIOS. This is because low-interaction honeypots can be easily verified if the attacker suspects that deception is employed. Use of high-interactive honeypots for deception is more fool-proof but consumes more computing and memory resources. Thus, smart-box helps in efficient and effective allocation of these honeypot resources by assessing the nature of an attack/attacker and re-routing the traffic to appropriate honeypots (similar to loading the deception scripts from the repository).

Logging tools and analyzer in the honeypot farm are responsible for generating AIOS. Based on this AIOS, the flow is either whitelisted and forwarded to the production server or blacklisted. If blacklisted, either automated patches (if available) are scheduled for execution or the administrator is alerted. In this step, AIOS helps the defender to develop an effective patch for the next recovery cycle while the unsuspected malicious actor stays busy playing with the honeypot. Thus, deception buys defender the time to design an effective recovery.

Since a system is "as secure as its weakest point", the need is to ensure that this solution not only provides survivability but is tamper-proof at all times. Since all modules in this design, such as proxy servers, the traffic redirection module, intrusion detection systems, etc. are connected to the same network, they are always susceptible to intrusions. Therefore, these modules need to be tamper-proof in order for the entire design to be tamper-proof. The various schemes discussed through Chapters 3-5 can be used for this purpose.

6.4 Discussion and Conclusion

The SWAT extension described above is to be used at the end-systems to effectively and efficiently identify and prevent against the zero-day attacks. It involves deception based prevention, detection and recovery techniques along with a module (smart-box) to customize deception in real-time for expedited profile development.

While designing any good survivability solution for a strong adversary model, assuming attacker's limitations is never wise. Therefore, a good deception should be non-verifiable [99]. Deception is difficult to create but easier to verify. For instance, when an attacker attempts to delete a file and a deceptive interface gives a positive deletion confirmation but does not delete it, the attacker can still verify if the file exits. Assuming that for a state s(t), an action χ is expected to have an effect ω. Generally, deception (like in honeypots) involves confirming that χ has been performed but the effect ω is never reflected in the system. If the attacker has a feedback loop to verify ω, deception can be easily identified. Therefore, either the feedback loop needs to be controlled so as to give the impression that ω exists or the feedback loop should be blocked for all regular users. An open and honest feedback loop only helps the attacker to figure out ways around deception by trial-and-error.

There are several implementation-specific challenges involved in the designing of an effective solution. For instance, designing proxy servers, re-routing the traffic, choosing the right IDS, etc. Designing an effective smart-box presents two interesting challenges. First, how to assess the nature of the traffic flow and second, how to map the AIOS to an appropriate honeypot in the farm. Designing an implementation of both these functions can benefit from the use of machine learning algorithms. Deceptions in honeypots can also be customized based on the parameters provided by the smart-box.

In the next chapter, we conclude this dissertation by briefly describing the work and discussing its effects on current survivability scenario, as well as, the possible directions to further this research.

Chapter 7

Discussion and Conclusion

This chapter presents a brief overview of the work presented in this dissertation, highlights the key research issues and identifies the areas for future work.

The increasing complexity of mission critical systems combined with the emergence of advanced persistent threats (APT) demand stronger mission survivability. This dissertation presents a survivability solution for this purpose called SWAT (Survivability in (cyber) Warfare against Advanced Threats). SWAT combines traditional survivability solutions such as prevention, detection, recovery and adaptation with deception techniques in order to strengthen mission survivability. It uses deception at each stage to surreptitiously gather information about suspicious events. This information is reported and processed stealthily. Any correlations are used in the development of attacker's intent, objectives and strategies (AIOS). SWAT also leverages strategical placement of honeypots near the end systems to improve system's resilience against zero-day attacks. Effective use of honeypots can further expedite AIOS development. AIOS is used in the real-time design and customization of targeted recovery procedures. Such AIOS-based recoveries are beneficial in two major ways. First, they are more selective and effective than generic recoveries and tamper-free processes do not suffer unnecessarily. This eventually leads to stronger mission survivability wherein only the tampered processes are revived and penalized. Second, surreptitious information gathering and reporting enables the recovery to repair more intruder access points at once. Generic and incremental recoveries are seldom effective against stealthy attackers who have established several access points throughout the system. In a way, generic recoveries provide valuable feedback to the intruders about system's defense and further help their future attack strategies without effectively blocking them out of the system. Targeted recoveries can avoid this effectively if implemented correctly.

Chapter 1 provides a detailed introduction to the survivability problems (especially due to APT) in existing mission critical systems. In addition, it also discusses the solution approach and provides a summary of the contributions made by this dissertation.

Chapter 2 presents a review of the literature relevant to the current survivability scenario. It also describes the recent instances of cyber attacks that can be categorized under APT. These instances are used to derive a generic APT-based attack model in Chapter 3. This chapter also discusses the role of deception in designing defense solutions for the cyber space. It is realized during this discussion that deception has long been used for defense in cyber scenarios or otherwise. However, its related ethical and legal issues are an ongoing topic of debate.

Chapter 3 describes a generic APT-based attack model. It then continues to define a generic centralized, fault-tolerant system whose survivability is targeted by advanced persistent attacks. A centralized version of SWAT is used to provide this system with strong survivability. Centralized SWAT involves the use of software-based tripwires that gather suspicious event information and stores it in the hardware in the form of integrity-signatures. The hardware component used for surreptitious and tamper-free storage of this integrity-signature uses the existing test-logic hardware. This preexisting hardware comes at no extra cost and hence makes centralized SWAT cost-effective. SWAT uses deception at each step for hiding or misrepresenting information to the attacker. The integrity-signature is used by a central authority to design a targeted recovery that can effectively recover the remote system from an advanced compromise. SWAT is lightweight in terms of its time and performance overheads and hence conserves the timeliness property of survivability as well. It does not have any application-specific dependencies and thus, its implementation has the potential to be application transparent.

Performance evaluation of centralized SWAT employs multi-step evaluation. Multi-step approach is required because there are no existing benchmarks for complex systems such the ones that employ SWAT. SWAT involves interactions between the various hardware and software components that are difficult to model for simulation. Furthermore, an experimental prototype development is also hard in this case because of the minor changes required at the hardware layer to the test-logic. Therefore, multi-step evaluation provides a combination of theoretical analysis, pilot system implementation and simulation in order to deliver more realistic and statistically accurate results. The evaluation of centralized SWAT shows promising results. Overall, SWAT is believed to provide strong survivability at low cost for mission critical applications.

The discussion of centralized SWAT covers most of the major issues except the following two. First, how to ensure that security monitors trusted to report integrity-signatures stay tamper-resistant at all times? For this purpose, a solution is presented in the Chapter 4. Second, how to encrypt the communication between nodes? Test-logic circuitry can be used only for storage but not processing. However, node-to-node verification requires the communication to be encrypted. Use of trusted platform module (TPM) is explored in Chapter 5 for this purpose.

Chapter 4 presents a scheme to ensure the tamper-resistance of critical components such as security monitors in SWAT. The solution described in this chapter consists of lightweight watchers (processes that monitor other processes) arranged in a cyclic configuration in a multi-core environment. A uni-core solution along the same lines had been proposed earlier in the works of Chinchani et al. [26]. This solution can have several topologies. Circulant digraph and adaptive topologies are discussed in detail. Circulant digraph topology provides good security whereas adaptive topology aims to provide a fine balance between security and performance overhead. The scheme organizes light-weight watchers in cycles with directed edges. These directed edges represent periodic checks for specific conditions such as if the monitored program is continuously running. Thus, the quality of tamper-resistance provided by this solution is implementation-dependent to some extent because the conditions defining *unexpected behavior* are decided and implemented by security practitioners. The watchers are expected to be lightweight allowing easy formal verification of code correctness. They are also expected to have low number of false positives because of the simplicity of checks. Any false negatives will most likely result from the developers missing to implement all the conditional checks effectively. The solution is lightweight in terms of time and performance overhead and thus, can be employed as part of a mission survivable solution.

Chapter 5 extends SWAT for effective and efficient operation against APT in a distributed environment. The basic idea is the same as in centralized SWAT, to buy defense enough time to figure out the precise nature of an attack (AIOS). This helps in the designing of targeted recovery procedures. Distributed SWAT, as in centralized SWAT, disguises the knowledge of detection from the attacker. This 'false assurance' prevents the attackers from switching to an aggressive strategy or contingency plan. Each node in distributed SWAT runs application replicas in lockstep but on separate and isolated virtual machines. They also employ IDS that surreptitiously report suspicious activities to the TPM (in the form of integrity-signatures). Nodes exchange sets of replica-results and their integrity-signatures for verification by other nodes. All the components of distributed SWAT that require tamper-resistance employ watchdog cyclic solution described in Chapter 4. Compromised replicas are not recovered immediately but are monitored for refining AIOS. Meanwhile, the intermediate good results are obtained from the tamper-free replicas. Security strength analysis of distributed SWAT shows that it is inherently secure by design. Simulations show the distributed SWAT to have low time and performance overhead. Strong security and tamper-resistance properties of distributed SWAT, along with its efficiency, could make it a good defense strategy against APTs for which no good solutions currently exist.

Chapter 6 describes a SWAT extension to be deployed at the end-systems to effectively and efficiently identify and prevent zero-day attacks. It involves deception based prevention, detection and recovery techniques along with a module (smart-box) to customize deception in real-time for expedited profile development.

SWAT employs deception extensively and at each step of its survivability design. However, there are several pitfalls that one should be aware about when it comes to implementing deception for real-world purposes. First, good deceptions should not be verifiable as discussed in Chapter 6. Hence, they should be either made non-verifiable or the feedback loop that can confirm their existence to normal users should be closed. Second, good deceptions are system and situation specific. Generic deceptions can be easily verified if they are implemented in different systems and are not customized appropriately. Third, there are several legal and ethical issues that hinder the widespread adoption of deception. Many of them have been discussed by researchers like Cohen [27] and Lakhani [69] over the years. Despite these, there is a visible trend of the increasing use of honeypots for defense in diverse systems.

Missions with long timelines (especially the ones that spread over months and years) are especially threatened by the APTs as discussed in Chapter 3. Any good solution developed for long-running missions should remain effective throughout the lifetime of the mission. In other words, the effectiveness of a security solution should not be time-dependent. The design of SWAT satisfies this requirement sufficiently.

SWAT provides strong survivability but makes certain assumptions in order to get a handle on the challenging problem of stealth attacks. For instance, it treats the cyber defender as a power player, being able to see and control all the parameters, while the attacker has no clue as to what is going on at the other end. In real world, however, such a power bias rarely exists. A deeper look at SWAT reveals the following strong assumptions – (a) security monitors such as the Tripwire can detect every suspicious incident on a system, (b) the detection information such as the integrity-signature is completely hidden from the adversary, (c) there are no false alarms, and (d) the attacker has absolutely no idea about the existence of a deceptive defense system in place. In reality, these assumptions will not hold true and the SWAT design should move beyond these assumptions in order to arrive at realistic solutions and tools. This means, among other things, adding more intelligence to the smart box and other modules that transform information collected into defensive reactions (such as recoveries). Therefore, the need is to reach a power balance between the attacker and the defender. For example, the attacker can be assumed to have the knowledge of deceit existence. This kind of power balance and some common knowledge can be formulated as a game between an attacker and the defender. Game theory has been used for a long time to model such interactions and researchers have already used game theory as a tool to analyze and provide strong network security, as surveyed in by Roy et al. [118]. Effort in this direction will help generalize the initial formulations and provide more insight into developing mechanisms and tools to handle the scourge of stealth attacks.

To conclude, SWAT identifies deception as the key to withstanding advanced attacks. All its constituent solutions do excellent on maintaining low time and performance overheads. In general, solutions based on deception can benefit from adopting game-theory and letting go of the assumption that the deception will always be hidden. Therefore, an important avenue that requires further exploration in SWAT is the adoption of game theory for real-time customization of deceptions.

Chapter 8

List of Abbreviations

Abbreviation	Expansion
AIOS	Attacker Intent, Objectives and Strategies
AV	Anti-Viruses
BIST	Built-In-Self-Test
C&C	Command and Control
COTS	Commercial-Of-The-Shelf
CPU	Central Processing Unit
CTMC	Continuous-Time Markov Chains
DFT	Design For Testability
DoS	Denial-Of-Service
FFT	Fast Fourier Transform
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IPS	Intrusion Prevention System
JTAG	Joint Test Action Group
LU	dense LU matrix factorization

Abbreviation	Expansion
PCI	Peripheral Component Interconnect
PCR	Platform Configuration Register
PE	Processing Element
PLC	Programmable Logic Controllers
SCADA	Supervisory Control And Data Acquisition
SIFT	Software Implemented Fault Tolerance
SMI	System Management Interrupt
SMM	System Management Mode
SOR	Jacobi Successive Over-relaxation
Sparse	Sparse Matrix multiplication
SWAT	Survivable framework for cyber Warfare against Advanced Threats
TCB	Trusted Computing Base
TDI	Test Data Input
TOCTOU	Time-Of-Check-to-Time-Of-Use
TPM	Trusted Platform Module
VM	Virtual Machine
WoV	Window of Vulnerability

Bibliography

[1] A. L. Williamson. Discrete Event Simulation in the Mbius Modeling Framework. *Master's Thesis, University of Illinois*, 1998.

[2] M. Abramovici and C. Stroud. BIST-based Test and Diagnosis of FPGA Logic Blocks. *IEEE Transactions on VLSI Systems*, 9:159–172, 2001.

[3] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein. A Comparative Analysis of Network Dependability, Fault-tolerance, Reliability, Security, and Survivability. *IEEE Communications Surveys and Tutorials*, 11:106–124, 2009.

[4] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. *Proceedings of the 14th USENIX Security Symposium*, 2005.

[5] apk. Interface Promiscuity Obscurity. *Phrack Magazine (article 10 of 15)*, 8, 1998.

[6] K. Askola, R. Puupera, P. Pietikainen, J. Eronen, M. Laakso, K. Halunen, and J. Roning. Vulnerability Dependencies in Antivirus Software. *Second International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, pages 273 –278, aug. 2008.

[7] K. M. M. Aung, K. Park, and J. S. Park. Survival of the Internet Applications: A Cluster Recovery Model. *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops*, 2:33, 2006.

[8] R. Automation. Arena Software Simulation. *http://www.arenasimulation.com*, 2000.

[9] S. Axelsson. Technical Report, Department of Computer Engineering, Chalmers University of Technology. *Intrusion Detection Systems: A Survey and Taxonomy*, 99, 2000.

[10] A. Azab, P. Ning, Z. Wang, X. Jiang, and X. Zhang. HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity. *17th ACM Conference on Computer and Communications Security (CCS)*, 2010.

[11] S. Bajikar. Trusted Platform Module (TPM) Based Security on Notebook PCs - White Paper. *Mobile Platforms Group, Intel Corporation*, 2002.

[12] S. Bake, N. Filipiak, and K. Timli. In the Dark: Crucial Industries Confront Cyberattacks. *McAfee Second Annual Critical Infrastructure Protection Report*, 2011.

[13] M. Banatre, A. Pataricza, A. Moorsel, P. Palanque, and L. Strigini. From Resilience-building to Resilience-scaling Technologies: Directions ReSIST. *NoE Deliverable D13. DI/FCUL TR 0728, Department Of Informatics, University of Lisbon*, 2007.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review (SOSP)*, 37(5):164–177, oct 2003.

[15] V. R. Basili and B. T. Perricone. Software Errors and Complexity: an Empirical Investigation0. *Communications of the ACM*, 27:42–52, January 1984.

[16] R. Baskerville. Information Warfare Action Plans for e-Business. *European Conference on Information Warfare and Security*, pages 15–20, 2004.

[17] L. E. Bassham and W. T. Polk. Threat Assessment of Malicious Code and Human Threats (NISTIR 4939). *National Institute of Standards and Technology Computer Security Division*, 1992.

[18] A. Bessani, H. Reiser, P. Sousa, I. Gashi, V. Stankovic, T. Distler, R. Kapitza, A. Daidone, and R. Obelheiro. FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery. *Proceedings of the ACM Middleware Companion*, 2008.

[19] L. Bridges. The Changing Face of Malware. *Network Security*, pages 17–20, 2008.

[20] V. Bukac, P. Tucek, and M. Deutsch. Advances and challenges in standalone host-based intrusion detection systems. *Trust, Privacy and Security in Digital Business (Springer Berlin / Heidelberg)*, 7449:105–117, 2012.

[21] Bulba and Kil3r. Bypassing Stackguard and Stackshield. *Phrack Magazine, 56*, 2000.

[22] M. Carvalho. A Distributed Reinforcement Learning Approach to Mission Survivability in Tactical MANETs. *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, pages 21:1–21:4, 2009.

[23] M. Carvalho, D. Dasgupta, and M. Grimaila. Mission Resilience in Cloud Computing: A Biologically Inspired Approach. *6th International Conference on Information Warfare and Security*, pages 42–52, 2011.

[24] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20:398–461, 2002.

[25] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys (CSUR)*, 41:15:1–15:58, 2009.

[26] R. Chinchani, S. Upadhyaya, and K. Kwiat. A Tamper-resistant Framework for Unambiguous Detection of Attacks in User Space Using Process Monitors. *First IEEE International Workshop on Information Assurance (IWIAS)*, pages 25–34, 2003.

[27] F. Cohen. Deception Toolkit. [online] 2001. *http://all.net/contents/dtk.html*.

[28] F. Cohen, D. Lambert, C. Preston, N. Berry, C. Stewart, and E. Thomas. A Framework for Deception. *Computers and Security (IFIP-TC11)*, 2001.

[29] E. Cole. Advanced Persistent Threat: Understanding the Danger and How to Protect your Organization. *Syngress*, 2012.

[30] M. K. Daly. The Advanced Persistent Threat. *Large Installation System Administration Conference (LISA)*, 2009.

[31] D. C. Daniel and K. L. Herbig. Strategic Military Deception. *Pergamon Press*, 1982.

[32] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Longstaff, and N. R. Mead. Survivability: Protecting Your Critical Systems. *IEEE Internet Computing*, 3:55–63, 1999.

[33] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson. A Survey of Rollback-recovery Protocols in Message-passing Systems. *ACM Computing Surveys (CSUR)*, 34:375–408, 2002.

[34] S. Embleton, S. Sparks, and C. Zou. SMM Rootkits: a New Breed of OS Independent Malware. *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, 2008.

[35] J. Eronen, J. Eronen, R. Puuper, E. Kuusela, K. Halunen, M. Laakso, and J. Rning. Software Vulnerability vs. Critical Infrastructure - a Case Study of Antivirus Software. *International Journal On Advances in Security*, 2(1), 2009.

[36] N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier - White paper. *Symantec Corporation, Security Response*, 2011.

[37] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1997.

[38] C. A. Fowler and R. F. Nesbit. Tactical Deception in Air-land Warfare. *Journal of Electronic Defense*, 1995.

[39] FreeBSD. Freebsd Problem Report Kern/29741. [online] 2002. *http://www.freebsd.org/cgi/query-pr.cgi?pr=kern/29741*.

[40] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a Virtual Machine-based Platform for Trusted Computing. *SIGOPS Operating System Review*, 37(5):193–206, oct 2003.

[41] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *ACMSymposiumonOperatingSystemsPrinciples*, 2003.

[42] D. Geer. Chip Makers Turn to Multicore Processors. *Computer*, 38:11–13, 2005.

[43] R. Geist and K. Trivedi. Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques. *Computer*, 23:52–61, 1990.

[44] K. K. Goswami, R. K. Iyer, and L. Young. DEPEND: A Simulation-Based Environment for System Level Dependability Analysis. *IEEE Transactions on Computers*, 46:60–74, 1997.

[45] R. Goyal, S. Sharma, S. Bevinakoppa, and P. Watters. Obfuscation of Stuxnet and Flame Malware. *Latest Trends in Applied Informatics and Computing*, 2012.

[46] W. Gragido and J. Pirc. Cyber Crime and Espionage: Seven Commonalities of Subversive Multivector Threats. *Syngress*, 2011.

[47] M. J. Gross. A Declaration of Cyber-War. [online] 2011. *http://www.vanityfair.com/culture/features/2011/04/stuxnet-201104*.

[48] A. Haeberlen, P. Kouznetsov, and P. Druschel. The Case for Byzantine Fault Detection. *Proceedings of the 2nd Conference on Hot Topics in System Dependability,*, 2, 2006.

[49] S. Hakimi and A. T. Amin. On the Design of Reliable Networks. *Networks*, 3:241–260, 1973.

[50] halflife. Bypassing Integrity Checking Systems. *Phrack Magazine (article 9 of 17)*, 7, September 1997.

[51] G. Hoglund. Malware Commonly Hunts Down and Kills Anti-virus Programs. [online] 2009. `http://fasthorizon.blogspot.com/2009/03/malware-commonly-hunts-down-and-kills.html`.

[52] A. Hrivnak. Host Based Intrusion Detection: An Overview of Tripwire and Intruder Alert. *SANS Institute InfoSec Reading Room*, 2002.

[53] X. Huangang. Building a Secure System With LIDS. [online] 2010. `http://www.lids.org/document/build_lids-0.2.html`.

[54] Intel. Teraflops Research Chip. [online]. `http://www.intel.com/content/www/us/en/research/intel-labs-teraflops-research-chip.html`.

[55] K. Jain and R. Sekar. A User Level Infrastructure for System Call Interception: A Platform for Intrusion Detection and Confinement. *Network and Distributed Systems Security Symposium (NDSS)*, 2000.

[56] D. G. Jr. Monopoly Considered Harmful. *IEEE Security and Privacy*, 1:14–17, 2003.

[57] Z. Kalbarczyk, R. Iyer, S. Bagchi, and K. Whisnant. Chameleon: a Software Infrastructure for Adaptive Fault Tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10:560–579, 1999.

[58] Z. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant. Chameleon: A Software Infrastructure for Adaptive Fault Tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10:560–579, 1999.

[59] A. Kapoor and R. Mathur. Predicting the Future of Stealth Attacks. *Virus Bulletin Conference*, 2011.

[60] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu. Social Network-based Botnet Command-and-control: Emerging Threats and Countermeasures. *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 511–528, 2010.

[61] S. M. Khan. Vulnerability Centric Exploitation Attempts in Open Source Software Systems. *International Journal of Computer Technology and Applications*, 3, 2012.

[62] G. H. Kim and E. H. Spafford. The Design and Implementation of Tripwire: a File System Integrity Checker. *Proceedings of the Second ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[63] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. SubVirt: Implementing Malware With Virtual Machines. *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 2006.

[64] K. J. Knappa and W. R. Boulton. Cyber-Warfare Threatens Corporations: Expansion into Commercial Environments. *Information Systems Management*, 23:76–87, 2006.

[65] C. Ko, T. Fraser, L. Badger, and D. Kilpatrick. Detecting and Countering System Intrusions Using Software Wrappers. *Proceedings of the 9th USENIX Security Symposium*, 2000.

[66] R. Kuhn and C. Johnson. Vulnerability Trends: Measuring Progress. *IT Professional*, 12:51–53, 2010.

[67] I. T. Lab. The Blue Pill Project. [online]. *http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html*.

[68] M. Labs. Protecting Your Critical assets, Lessons Learned From " Operation Aurora". *Technical report*, 2010.

[69] A. D. Lakhani. Deception Techniques Using Honeypots. *MSc Thesis, ISG, Royal Holloway, University of London*, 2003.

[70] C. Lam. Hadoop in Action. *Manning Publications Co.*, 2010.

[71] M. Laurean, C. Maziero, and E. Jamhour. Protecting Host-based Intrusion Detectors Through Virtual Machines. *Computer Networks*, 51:12751283, 2007.

[72] T. Lawless. St Michael: Detection of Kernel Level Rootkits. [online] 2009. *http://sourceforge.net/projects/stjude*.

[73] A. Lee and P.-M. Bureau. The Evolution of Malware. *Virus Bulletin Conference*, pages 8–10, 2007.

[74] C. E. Leiserson and I. B. Mirman. How to Survive the Multicore Software Revolution (or at Least Survive the Hype). *Cilk Arts Inc.*, 2008.

[75] J. Lemon. Kqueue: A Generic and Scalable Event Notification Facility for Freebsd. *BSDCon*, 2000.

[76] J. G. Levine, J. B. Grizzard, and H. L. Owen. Using Honeynets to Protect Large Enterprise Networks. *IEEE Security and Privacy*, 2:73–75, 2004.

[77] S. Levitan and D. M. Chiarulli. Massively Parallel Processing: It's Dj Vu all Over Again. *46th ACM/IEEE Design Automation Conference (DAC)*, pages 534–538, 2009.

[78] X. Lin, M. Zhu, and R. Xu. A Framework for Quantifying Information System Survivability. *Third International Conference on Information Technology and Applications*, 2:552–555, 2005.

[79] J.-C. Liou. When the Software Goes Beyond its Requirements – A Software Security Perspective. *Proceedings of the 2012 Ninth International Conference on Information Technology - New Generations*, pages 403–408, 2012.

[80] P. Liu, W. Zang, and M. Yu. Incentive-based Modeling and Inference of Attacker Intent, Objectives, and Strategies. *ACM Transactions on Information and System Security (TISSEC)*, 8, 2005.

[81] Y. Liu and K. Trivedi. A General Framework for Network Survivability Quantification. *Proceedings of 12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems together with 3rd Polish-German Teletraffic Symposium*, 2004.

[82] P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, 2001.

[83] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A Real-time Intrusion-detection Expert System (IDES). *SRI International, Computer Science Laboratory*, 1992.

[84] M. Ramilli and M. Bishop. Multi-stage delivery of malware. *5th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 91–97, 2010.

[85] Mandiant Corporation. APT1: Exposing One of China's Cyber Espionage Units. [online] 2013. *http://intelreport.mandiant.com*.

[86] R. Masood, U. Um-e-Ghazia, and Z. Anwar. SWAM: Stuxnet Worm Analysis in Metasploit. *Frontiers of Information Technology (FIT)*, pages 142–147, 2011.

[87] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2008.

[88] W. L. McGill. Defensive Dissuasion in Security Risk Management. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2009.

[89] W. L. McGill, B. M. Ayyub, and M. Kaminskiy. Risk Analysis for Critical Asset Protection. *Blackwell Publishing Inc*, 27:1265–1281, 2007.

[90] B. Meyer. Object-oriented software construction. *Prentice Hall*, 1988.

[91] B. Miller and R. Pozo. Scimark 2.0 Benchmark. [online] 2004. http://math.nist.gov/scimark2.

[92] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A Distributed Parallel Simulator for Multicores. *6th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, 2010.

[93] S. Moitra and S. Konda. Simulation Model for Managing Survivability of Networked Information Systems. *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2000-TR-020, 2000*, 2000.

[94] J. Molina and M. Cukier. Evaluating Attack Resiliency for Host Intrusion Detection Systems. *Journal of Information Assurance and Security*, 4:1–9, 2009.

[95] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-core Systems. *Proceedings of 16th USENIX Security Symposium*, 18:1–18, 2007.

[96] K. Munro. Deconstructing Flame: the Limitations of Traditional Defences. *Computer Fraud and Security*, pages 8–11, 2012.

[97] B. S. Murphy. Deceiving Adversary Network Scanning Efforts Using Host-Based Deception. 2009.

[98] E. Nakashima and J. Pomfret. China Proves to be an Aggressive Foe in Cyberspace. [online] 2009. http://www.washingtonpost.com/wp-dyn/content/article/2009/11/10/AR2009111017588.html.

[99] V. Neagoe and M. Bishop. Inconsistency in Deception for Defense. *Proceedings of the Workshop on New Security Paradigms*, 2007.

[100] Y. Nishikawa, M. Koibuchi, M. Yoshimi, A. Shitara, K. Miura, and H. Amano. Performance Analysis of ClearSpeed's CSX600 Interconnects. *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 203–210, 2009.

[101] C. Null. New Malware Attack Laughs at Your Antivirus Software. [online] 2010. http://old.news.yahoo.com/s/ytech_wguy/20100510/tc_ytech_wguy/ytech_wguy_tc1985, 2010.

[102] OPSWAT. Security Industry Market Share Analysis. [online] 2012. *http://www.opswat.com/sites/default/files/OPSWAT-market-share-report-march-2012.pdf*.

[103] I. V. Paputungan, A. Abdullah, and L. T. Jung. Critical Service Recovery Model for System Survivability. *Proceedings of the 9th WSEAS Inernational Conference on Mathematical and Computational Methods in Science and Engineering*, pages 246–252, 2007.

[104] J. Park and P. Chandramohan. Static vs. Dynamic Recovery Models for Survivable Distributed Systems. *Proceedings of the 37th Hawaii International Conference on System Sciences* , pages 5–8, 2004.

[105] R. R. Patel and C. S. Thaker. Zero-Day Attack Signatures Detection Using Honeypot. *International Conference on Computer Communication and Networks (CSI- COMNET)*, 2011.

[106] D. Patterson and H. J. Computer Organization and Design: The Hardware/Software Interface. *Morgan Kaufmann*, 1994.

[107] B. Payne, M. Carbone, M. Sharif, and W. Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. *IEEE Symposium on Security and Privacy*, pages 233–247, 2008.

[108] P.M.Curtis. Maintaining Mission Critical Systems in a 24/7 Environment. *John Wiley & Sons, Inc.,*, 2007.

[109] Pongor , György. OMNeT: Objective Modular Network Testbed. *Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 323–326, 1993.

[110] G. Portokalidis and H. Bos. SweetBait: Zero-Hour Worm Detection and Containment Using Low- and High-Interaction Honeypots. *Science Direct*, 51:1256–1274, 2007.

[111] D. Powell and R. S. (Editors). Conceptual Model and Architecture of MAFTIA. [online] 2003. *MAFTA Project Deliverable D21,* *http://www.maftia.org*.

[112] N. Provos and T. Holz. Virtual Honeypots: From Botnet Tracking to Intrusion Detection. *Addison-Wesley*, 2008.

[113] M. T. Qassrawi and H. Zhang. Deception Methodology in Virtual Honeypots. *Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC)*, 2:462–467, 24–25, 2010.

[114] N. A. Quynh and Y. Takefuji. A Novel Approach for a File-system Integrity Monitor Tool of Xen Virtual Machine. *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, pages 194–202, 2007.

[115] Rajkumar Buyya and Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14:1175–1220, 2002.

[116] K. A. Repik. Defeating Adversary Network Intelligence Efforts with Active Cyber Defense Techniques. *Graduate School of Engineering and Management, Air Force Institute of Technology*, 2008.

[117] N. C. Rowe and H. S. Rothstein. Two Taxononmies of Deception for Attacks on Information Systems. *Journal of Information Warfare*, 3:27–39, 2004.

[118] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Qishi. A survey of game theory as applied to network security. *43rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, 2010.

[119] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. L. Griffin, and S. Berger. sHype: Secure Hypervisor Approach to Trusted Virtualized Systems. *Research Report RC23511, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA*, 2005.

[120] T. Sandler and H. E. Lapan. The Calculus of Dissent: An Analysis of Terrorists' Choice of Targets. 76:245–261, 1998.

[121] G. Schryen, M. Volkamer, S. Ries, and S. M. Habib. A Formal Approach Towards Measuring Trust in Distributed Systems. *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011.

[122] H. Schwetman. CSIM: a C-based Process-oriented Simulation Language. *Proceedings of the 18th Conference on Winter Simulation*, pages 387–396, 1986.

[123] M. Shahzad, M. Shafiq, and A. Liu. A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles. *34th International Conference on Software Engineering (ICSE)*, pages 771–781, 2012.

[124] T. Shi, J. Zhao, X. Yin, and J. Wang. Research on Telecommunication Switching System Survivability Based on Stochastic Petri Net. *3rd International Conference on Innovative Computing Information and Control*, page 413, 2008.

[125] M. Smart, G. R. Malan, and F. Jahanian. Defeating TCP/IP Stack Fingerprinting. *Proceedings of the 9th conference on USENIX Security Symposium*, 9:17, 2000.

[126] A. Smith and N. Toppel. Case study: Using Security Awareness to Combat the Advanced Persistent Threat. *Thirteenth Colloquium for Information Systems Security Education*, 2009.

[127] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo. Resilient Intrusion Tolerance Through Proactive and Reactive Recovery. *Proceedings of the 13th IEEE Pacific Rim International Symposium on Dependable Computing*, page 373380, 2007.

[128] L. Spitzner. Honeynet Project, Know Your Enemy: Defining Virtual Honeynets. [online] 2008. *http://www.honeynet.org/papers/virtual*.

[129] A. Srivastava and J. Giffin. Recent Advances in Intrusion Detection. *Lecture Notes in Computer Science, Springer Berlin / Heidelberg*, 5230:39–58, 2008.

[130] M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the Reliability of Commodity Operating Systems. *ACM Transactions on Computer Systems (TOCS)*, 23:77–110, 2005.

[131] Symantec. CoreGuard Antivirus. [online] 2009. *http://www.symantec.com/security_response/writeup.jsp?docid=2009-043009-2213-99*.

[132] Symantec. Trojan.FakeAV [online] 2007. *http://www.symantec.com/security_response/writeup.jsp?docid=2007-101013-3606-99*.

[133] C. Tankard. Advanced Persistent Threats and how to Monitor and Deter Them. *Network Security*, pages 16–19, 2011.

[134] Trusted Computing Group, Incorporated. TCG Software Stack (TSS) Specification Version 1.2. 2007.

[135] S. Tzu. The Art of War (Translated by James Clavell). *Dell Publishing, New York, NY*, 1983.

[136] A. Vasudeva, A. K. Sharma, and A. Kumar. Saksham: Customizable x86 Based Multi-Core Microprocessor Simulator. *First International Conference on Computational Intelligence, Communication Systems and Networks*, pages 220–225, 2009.

[137] D. Wagner and P. Soto. Mimicry Attacks on Host-based Intrusion Detection Systems. *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.

[138] J. Wang, K. Sun, and A. Stavrou. An Analysis of System Management Mode (SMM)-based Integrity Checking Systems and Evasion Attacks. *Depart of Computer Science, George Mason University*, 2011.

[139] J. Wang, H. Wang, and G. Zhao. ERAS - an Emergency Response Algorithm for Survivability of Critical Services. *First International Multi-Symposiums on Computer and Computational Sciences*, 2:97–100, 2006.

[140] S. J. Wang and A. Ghosh. HyperCheck:A Hardware-Assisted Integrity Monitor. *13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2010.

[141] D. Watson, M. Smart, G. R. Malan, and F. Jahanian. Protocol Scrubbing: Network Security Through Transparent Flow Modification. *IEEE/ACM Transactions on Networking*, 12:261–273, 2004.

[142] Websense Security Labs. Fake Input Method Editor(IME) Trojan. [online] 2010.

[143] J. Wensley. SIFT: Software Implemented Fault Tolerance. *Proceedings of Fall Joint Computer Conference (AFIPS)*, 41:243–253, 1972.

[144] R. Wichmann. Samhain: Distributed Host Monitoring. [online] 2006. `http://www.la-samhna.de/samhain`.

[145] R. Wojtczuk and J. Rutkowska. Xen 0wing Trilogy. *Black Hat Conference*, 2008.

[146] J. Yuill, D. Denning, and F. Feer. Using Deception to Hide Things from Hackers: Processes, Principles, and Techniques. *Journal of Information Warfare*, pages 26–40, 2006.

[147] G. Zhao, H. Wang, and J. Wang. A Novel Quantitative Analysis Method for Network Survivability. *First International Multi-Symposiums on Computer and Computational Sciences*, 2:30–33, 2006.

[148] Y. Zuo and B. Panda. Unifying Strategies and Tactics: A Survivability Framework for Countering Cyber Attacks. *IEEE International Conference on Intelligence and Security Informatics*, pages 119–124, 2009.