

Comparative Analysis of Intrusion-Tolerant System Architectures

Quyen Nguyen¹ and Arun Sood^{1,2}

International Cyber Center and Department of Computer Science

¹George Mason University, Fairfax, VA 22030

²SCIT Labs, Inc, Clifton, VA 20124

{qnguyeng, asood}@gmu.edu

Abstract

Today, institutions want to build open systems and provide services to the public via the Internet. Such systems would potentially expose security vulnerabilities, and become susceptible to attacks. Therefore, security is critical in order to ensure confidentiality, integrity, and availability for system data and services. With increasing sophistication of security attacks the protection of open system is more challenging. Intrusion tolerance should be part of the overall defense in-depth security solution. In this paper, we will study and compare different approaches to intrusion-tolerant system architectures, focusing on three different lines of approach. The case study of an Open Archival Information System will be used to illustrate the security features of those architectures in the face of malicious attacks. We also include a qualitative and comparative analysis with respect to confidentiality, integrity, availability, and data ex-filtration.

Keywords: Intrusion Tolerance, detection-based, recovery-based.

1. Introduction

As computer systems are opened to large populations of users via web services on the public Internet, the task of protecting them becomes more daunting. Such systems would potentially expose more vulnerabilities than before. Furthermore, security attacks are becoming more sophisticated, and current intrusion prevention and detection are simply inadequate. We conclude that intrusion tolerance systems (ITS) should be part of the solution for securing computer information systems; a more robust solution should borrow from the architectures discussed in this paper.

This paper is organized as follows. In Section 2, we propose a taxonomy for ITS architectures and map existing systems to the taxonomy categories. Section 3 describes the analysis methodology with attack model examples and a case study for illustration. Sections 4 through 6 are devoted to the study and qualitative analysis of three typical ITS architectures. The paper summarizes the comparison in Section 7 with a conclusion in Section 8.

2. Overview

2.1. ITS Characteristics

The ITS architectures discussed in the literature are often based on the principles of fault-tolerance. In order to fend off malicious faults, ITS utilizes techniques of redundancy, diversity, and reconfiguration of the services, components, and servers to ensure that the system will continue to provide services even in the presence of an intrusion. Overall the ITS architecture objective is to remove unwanted intrusions and restore the system to its normal state.

2.2. ITS Taxonomy

For ITS architectures, we propose a simple taxonomy with four categories:

1. Detection-triggered
2. Algorithm-driven
3. Recovery-based
4. Hybrid

Using the above approach, we classify ITS architectures, several of which were part of the OASIS project funded by DARPA [1].

Detection-triggered. These ITS architectures share the concepts of building multiple levels of defense for increasing system survivability. Most of them rely on intrusion detection that triggers recovery mechanisms. SITAR (Scalable Intrusion-Tolerant Architecture) [1]

is an ITS architecture aimed at protecting services provided by COTS servers against known and unknown external attacks, without modifying the servers. SITAR's approach is based on two capabilities: one to detect compromises, and the other to perform adaptive reconfiguration of the compromised servers. An attack only succeeds if it can bypass three levels of detection and testing,

For DPASA [5], the main design principles are to fortify the protection of assets via multiple zones and layers to contain external attacks. The host responsible for managing security defense lies in the innermost zone, while proxies are placed further out to intercept incoming traffic. The architecture also includes network-based IDS that detect intrusion and generate alerts.

The Willow Architecture [6] consists of multiple mechanisms to detect malicious faults, analyze system vulnerabilities, and perform reconfiguration in a distributed computing environment. Distributed intrusion detection sensors monitor application hosts.

Focusing on Database Systems, Peng Liu et al. [1] proposed schemes to contain, eliminate, and repair damage caused to data by intrusions. All these schemes have a detection layer that alerts repairing components.

DIT [9] is an adaptive architecture composed of a cluster of mediating proxies, and a monitor system capable of alerting in case of intrusion. The cluster of proxies can also detect anomalies based on an agreement protocol.

Similarly, HACQIT [10] augments its intrusion detection capability provided by IDS software packages via the usage of an agreement protocol variant. The approach of ITSI [11] is based on detecting intrusion at the network layer.

The effectiveness of Detection-based systems is measured by the difficulty encountered by an attacker to penetrate multiple defense levels without being detected.

Algorithm-driven. Systems in this category put emphasis on algorithms such as Voting algorithm, Threshold Cryptography, and Fragmentation-Redundancy-Scattering (FRS) to harden their resilience.

PASIS [1] employs threshold secret sharing scheme to encode and disperse data in survivable storage systems. Increasing intrusion tolerance requires increasing the number of replicas and threshold numbers.

The case of MAFTIA [2] is more subtle. Although, MAFTIA incorporates IDS sensors, the focus is not on

IDS capability instead, it relies on Authority and Transaction Management services that are developed on top of common services providing voting algorithms and k-threshold cryptography. Like MAFTIA, ITUA [12] is a middleware that aims to adaptively protect applications at the object level by using protocols for group communications and cryptography. Intrusion tolerance for CORBA middleware is based on the principles of FRS, Threshold Cryptography, and Voting algorithms [1]. Cryptography techniques, challenge-response protocols, and k-security models are adapted to provide intrusion tolerance for wireless sensor networks [13]. In [14], the authors showed that software can be secured by applying secret sharing to compiler techniques.

Recovery-based systems stand out from other categories by assuming that as soon as a system comes online, it is compromised; therefore, periodic restoration to a known good state is necessary. Software Rejuvenation was proposed in [15] to limit impact of software faults and aging; thus cluster survivability is increased. The authors demonstrated the benefits of this approach and analyzed the rejuvenation rate by means of stochastic process modeling.

The main idea of SCIT (Self-Cleansing Intrusion Tolerance) [3] is to have a group of servers with identical services, but with some diversity. Within this group of servers, round-robin cleansing will be performed to restore to the pristine image. The FOREVER service consists of removing faults via proactive reconfiguration implemented by robust underlying components [16]. Implementation of proactive recovery based on Hypervisor virtualization is reported in [17].

Hybrids. Some systems optimize efficiency and survivability by having hybrid solution such as the scheme proposed by Sousa et al. [7]; periodic system rejuvenation is combined with a "reactive recovery". This recovery is triggered when the perceived threat goes beyond a tolerable threshold that can affect the correct working of the system. COCA [1] is also a hybrid since it uses threshold cryptography and proactive secret-sharing.

Finally, while the proposed taxonomy is ITS architecture driven, it is possible to construct other taxonomies. For instance, a taxonomy based on targets of attacks which are to be protected and hardened by the intrusion tolerance scheme. Such a target can be an application, a middleware, a backend database system, or a network infrastructure.

In the following sections, we compare three representative architectures, namely SITAR [1], MAFTIA [2], and SCIT [3] by analyzing qualitatively how they operate to achieve intrusion tolerance.

3. Qualitative Analysis Methodology

To make our analysis more focused, we consider the use case of a large digital archive library system with the following functional requirements:

Producers submit electronic records into Archival Storage.

Consumers search and access records from Archival Storage.

Administrators schedule and monitor record ingest process. They also perform transformations of assets for electronic preservation. Asset metadata are stored in databases.

For the system to be accessible via the Internet, applications and web services are deployed on application servers connected to backend database and mass storage.

For the analysis, the following measures are used:

(CS) Confidentiality. A Consumer tries to gain unauthorized access to privacy data.

(IS) Integrity. A hacker tries to corrupt records and/or associated metadata.

(AS) Availability. A hacker tries to affect the system availability such as DoS attack.

(ES) Data Ex-filtration. A hacker tries to extract massive data out of the system. For a large scale archival system, this compromise is of high severity.

We evaluate the following aspects of an ITS architecture: (a) effectiveness to protect against malicious faults from the perspectives of three pillars of security; (b) differentiated paradigm to approach intrusion tolerance; (c) performance impact in terms of latency, and (d) ease of integration into application systems.

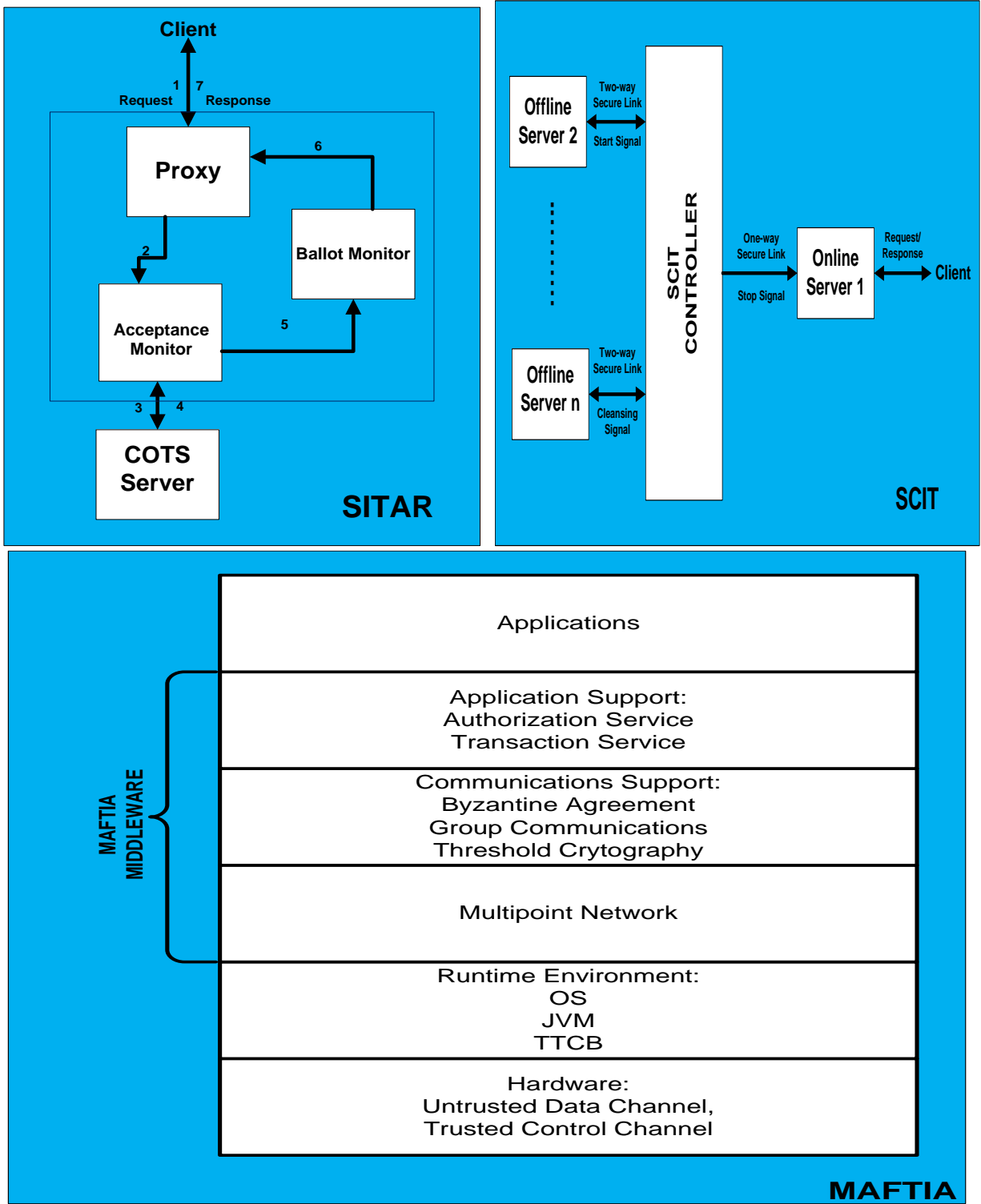


Figure 1. System Architectures of SITAR, MAFTIA, and SCIT.

4. Detection-triggered Approach

4.1. Description

Functional Architecture. Representative of the detection-triggered approach, SITAR is composed of five components, which are deployed in front of the servers to be protected as shown in Figure 1: Proxy Server, Acceptance Monitor (AM), Ballot Monitor, Adaptive Reconfiguration Module (ARM), and Audit Control (AC).

The Proxy Server is between all requesting clients and the COTS servers with same functionality but different implementations. After passing through the proxies, incoming client requests are validated by the AM, where acceptance validation is executed by the Test Engine in accordance with rules pre-configured in a Security Policy database. Invalid requests are not forwarded to the COTS servers; invalid responses from the COTS will trigger reconfiguration.

Upon receiving the responses from the AM instances (step 5 in Figure 1), BM adjudicates the final and correct response before sending to the Proxy (step 6) to forward to the client (step 7), by running the Voting Algorithm based on a simple majority, or a more sophisticated Byzantine agreement. BM calculates the transformations of the responses before the voting, using different checksum algorithms, depending on the level of security.

ARM is the centralized control to perform “adaptive” reconfiguration for a given security threat level.

Using log analytics, the AC facilitates the protection of SITAR components against compromises.

4.2. Analysis

4.2.1. Case Studies

Application. In order to apply SITAR for the OAIS case study, all we need to do is to deploy SITAR components in front of the OAIS system so that all requests and responses have to go through SITAR.

Scenario (CS). An unauthorized hacker attempts to access privacy data by one of the following ways:

- (CS1) She may obtain directory listing, path revealing, and file disclosure that could lead to privacy data by exploiting software vulnerabilities that exist in web-based applications [1]. Since these vulnerabilities exhibit known patterns in the malicious requests’ URLs, SITAR can incorporate rules to block these attacks.

- (CS2) In case she overcomes the above validation, AM can check the response(s) from the web server(s) via its Access Checker [2] to verify the access privilege of a directory or file against the access rules stored in the security policy database and reject the response.
- (CS3) She may impersonate an authorized user, by (a) snooping on the communication line, or (b) getting the access control list stored in the system. But both techniques would require breaking cryptographic keys used in communication and password storage. These two features are in the area of prevention and system hardening, and considered as outside of SITAR architecture.

Scenario (IS). A hacker can corrupt data in the OAIS system by uploading a file into a directory and exploiting URL related vulnerabilities. In this case, SITAR can stop the attack by using the same testing techniques used for the scenario (CS). Assume that she was successful in evading the acceptance testing and has uploaded her own file into the system. The impact could be web defacement. Now, if a normal user tries to access the corrupted file(s), then the response from the web server would not pass the Code Testing by AM [1]. Indeed, the checksum of the corrupted file will not match with that being stored in AM.

Scenario (AS). As pointed out in [1], the Timing Test, which is a kind of “reasonableness” test, can detect a DoS attack, because the web servers are so overloaded that response times will exceed the expected values.

Scenario (ES). By running reasonableness tests based on the size of a transfer, SITAR can detect an (ES) attack. In fact, we can configure the maximum size limit of data that are allowed to be transferred out of the data repository. However, this may not prevent data ex-filtration with low rate but long duration.

4.2.2. Discussion

Effectiveness. In general, SITAR is quite effective in surviving malicious faults. The architecture offers three lines of defense: validation of incoming requests, acceptance testing of outgoing responses, and majority balloting. The latter mechanism has the benefit that the system can still function, even though an instance of a COTS server is compromised, and undergoes reconfiguration. Balloting helps the system to be tolerant to unknown attacks, while acceptance testing detects known intrusion patterns.

As the case studies above exhibit, the architecture relies heavily on AM for detecting intrusions. Some acceptance tests, such as the checksum test for data

integrity or timing test for detecting response time degradation, are generic and can be applied to various types of applications. But, other tests such as URL pattern tests are specific to a particular application, so that provisioning them in the security policy database is quite a challenge. Moreover, violation rules have to be known in advance. So, monitoring the updates of vulnerability databases (e.g. National Vulnerability Database of <http://cve.mitre.org>, or BugTraq of <http://www.securityfocus.com>) should be an integral part of the maintenance.

The tunable parameter of SITAR is the number N of the COTS servers with same functionalities. N is also the number of AM instances, each of which is configured to test a COTS server.

Reconfiguration is performed automatically by the controlling module, which orchestrates all components in a consistent manner to achieve the desired quality of service level. Reconfiguration can be adapted to the current threat to increase the intrusion tolerance level. The degree of intrusion tolerance is tunable according to a cost versus tolerance trade-off.

Performance. Due to the introduction of two extra hops and processing between the originator of the requests and the application server, the service time will incur additional latency. This latency can increase if more sophisticated hashing algorithms are used in the ballot monitors, or more testing is performed in AM.

Integration. Integrating SITAR components to a protectee-system is tantamount to fronting it with a kind of proxy. Some network interface configuration is required, but there is no software change on the application side. The major effort lies in provisioning security rules.

5. Algorithm-driven Approach

5.1. Description

Basic Concepts. MAFTIA is built on the concept of Trust and Trustworthiness relationship. The degree with which a component X satisfies set of properties P is defined to be its *trustworthiness* (X). Given two components X and Y , X *trust* Y means that X accepts that failures of Y will compromise X . In a good design, we want that degree that X *trust* Y be less than *trustworthiness* (X).

With this concept, the architecture builds layers of increasing degree of trustworthiness, starting with trusted components in hardware hosts and networks.

Functional Architecture.

The MAFTIA middleware in Fig. 1 consists of services forming a runtime platform for applications. The intrusion tolerance of MAFTIA is based mostly on algorithms implemented in these services. MAFTIA itself runs on native OS, JVM, and a tamper-proof appliance board for TTCB (Time Trusted Computing Base).

At the network level, the *Multipoint Network* allows hosts to communicate in the trusted control channel.

The *Communication Support Services* are essentially protocols to ensure secure and robust communication between components. Examples are Byzantine agreement protocol and Cryptosystem with secret sharing. The Cryptosystem is characterized by a threshold T : the secret is shared between N ($N > T$) participants, and it requires $T+1$ valid shares to decrypt a message.

Two important *Activity Support Services* are *Authorization Service (AS)* and *Transaction Service (TS)*. The AS provides access control scheme at the macro level with the Authorization Server controlling multiple participants and the local host level with the *Reference Monitor (RM)*.

5.2. Analysis

5.2.1. Case Studies

Application. OAS applications need to be implemented on top of MAFTIA middleware (Applications layer in Figure) in order to utilize its intrusion tolerance features, especially the two prominent Authorization and Transaction services.

Scenario (CS). A hacker can access privacy data for which she is not authorized. Three scenarios can happen:

- (CS1) The Authorization Service has the RM component that resides locally on each host to control resources and objects local to that host. Hence, the exploitation of web server vulnerabilities based on URL patterns will not succeed.
- (CS2) The hacker may subvert the Authorization Service which controls access at the macro level. However, thanks to the secret-sharing cryptosystem, the subversion is successful only if k

authorization sites and k' secret shares are compromised. Below those thresholds k and k' , the attacks will be detected, which will trigger the recovery of the compromised components.

- (CS3) The hacker can realize the impersonation of an authorized user by acquiring the smartcard and guessing the PIN. But, it was argued in [2] that she cannot subvert the host authorization process thanks to the trusted Java card.

Scenario (IS).

- (IS1) Uploading a file into a directory can be prevented by the Authorization Service with its local component as discussed in the previous scenario (CS). Thus, malicious attempts to deface the website by using this line of attack can be blocked.
- (IS2) The hacker tries to compromise an ongoing transaction which involves creating or updating metadata in the OASIS system. Thanks to the redundancy of the Transaction Managers and their secure group communication service, the corruption of metadata will be detected and restored to the correct state. The Byzantine agreement protocol implemented in the Transaction Managers will help detecting compromised instances, while the transactional operation can still be processed with the correct data, as long as the number of compromises is less than the tolerance threshold.

Scenario (AS). The Authorization and Transaction services are not designed to fend off a DoS attack. The authors of MAFTIA have proposed a distributed IDS service consisting of redundant sensors to detect a DoS attack [2].

Scenario (ES). One may argue that the methods used to detect compromises in the (CS) scenario can also prevent an (ES) attack. But, given the specificity of an (ES) attack which focuses on the large amount of data transfer, it is not clear how MAFTIA can detect and contain loss in this scenario.

5.2.2. Discussion

Effectiveness. MAFTIA approach to intrusion tolerance is attractive because of the following reasons:

- The architecture is based on an interesting concept of Trust and Trustworthiness relationship, which helps to build layers of security services forming a top-to-bottom fortress to protect applications.
- The approach offers a complete solution with an entire stack for ITS from local host machines, network devices and protocols, OS extensions, group protocols, and other trusted security services.

Due to its completeness, one could argue that the architecture is quite complex.

- For its detection capabilities, MAFTIA relies on distributed protocols for voting and secret sharing. These support protocols are buttressed by trusted hardware and runtime environment at lower layers.

The tunable parameters for a MAFTIA architecture are the thresholds used in the Byzantine agreement protocols and the secret-sharing cryptosystem. Higher values of these thresholds will increase the resilience of the system.

Reconfiguration. Two actions are used to recover from error due to faults, malicious and accidental [2]. First, the error has to be contained so that it cannot be propagated to other members participating in a transaction. Second, the recovery of the corrupted member is performed by reading the states of other stable members.

Performance. The additional latency of the service time stems from the use of multiple participants in the Byzantine agreement protocol and secret-sharing cryptosystems.

Integration. To use MAFTIA intrusion tolerance services, an application must be developed on top of its middleware using its specified APIs and protocols [2]. This integration method appears somehow intrusive. Hence, the integration cost can be higher, in exchange for a tighter and more complete integration with a trusted architecture.

6. Recovery-based Approach

6.1. Description

Basic Concepts. The recovery-based SCIT model is applicable to servers that are open to the Internet, such as Web, and DNS servers[4]. Due to round-robin cleansing, at any point in time, a server in a SCIT cluster can have one of the three states: offline cleansing, online primary, and online backup. The duration that a SCIT server is exposed to the Internet is called its Exposure Time.

Functional Architecture. The architecture is simple, and does not rely on intrusion detection. Figure1 depicts the Controller with one online node serving incoming transactions, one node ready to be put online, and another in cleansing mode. Clearly, the core component of SCIT is the Central Controller which manages server rotation in and out of the cleansing mode. The server rotation follows an algorithm, which

takes into account the group cardinality, the cleansing cycle time of a server, and the number of required online servers. Implementation of SCIT controller algorithm is based on virtualization technology. The interfaces between the Controller and the group of servers to be protected are trusted.

6.2. Analysis

6.2.1. Case Studies

Application. For SCIT to protect the OAIS system, we need to instantiate multiple web servers, and connect those servers to the Central Controller.

Scenario (CS). Since there is no detection component in its architecture, a pure SCIT-based system is not hardened in the face of attacks that can impact data confidentiality. However, the chance for this kind of attack is limited by the duration that a server instance is exposed online.

Scenario (IS). Similarly, it is not clear how the system can protect integrity of the system in “theoretical” real-time model, i.e. a model with attacks coming as a continuous stream. In reality, intrusions are discrete events, separated from each other by some time interval. Ensuring data integrity in an intrusion tolerant system means that the system has the capability to restore back to pristine data and state. In this case, the system must have a tamper-proof storage that plays the role of master data. Thus, the master storage is trusted and used in the restoration of corrupted web servers. Moreover, the use of a time-based signature system whose cycle corresponds with the rotation time makes it harder for an intruder to break the cryptosystem in order to get to the data. How SCIT-based web server is tolerant in the case of web defacing attacks is discussed in [4]; and it is conceivable that the cleansing approach can be extended to a generic case where data integrity is at risk. However, the architecture can limit the possibility of a SQL injection attack, but cannot detect and stop it on the spot.

Scenario (AS). As it is currently designed, it is not clear how SCIT-based systems can still provide services with required quality of service when there is a DoS attack, notably in the case of a flooding attack. However, the counter-argument is that the duration of a DoS attack is limited to the web server instance which is online at the moment. SCIT recovers from failure every cycle, and once the SCIT self-cleansing starts the peer instance, the DoS attack will be no longer in effect.

Scenario (ES). The time-based paradigm enables SCIT to contain an (ES) compromise. The smaller the exposure window, the smaller is the loss of data due to an (ES) attack. In fact, this control can even be quantified, and hence managed according to some risk acceptance policy.

6.2.2. Discussion

Effectiveness. As essentially a recovery-based approach, SCIT differs greatly from the other two, since it doesn’t have an intrusion detection component. The strategy is based on the assumption that there would remain vulnerabilities in the system, and successful attacks are inevitable. So, the effectiveness of SCIT architecture cannot be analyzed qualitatively in terms of vulnerabilities. Instead, the degree of intrusion tolerance should be quantitatively evaluated in the context of how much the system can limit damages caused by malicious attacks and how much service availability it offers to the users [8]. The authors of SCIT have pointed out that nothing prevents a SCIT-based system to add architectural constructs to adapt to the particular requirements of an application. Examples can be found in papers which describe SCIT architectures for DNS, and Web Servers.

With the combination of a hardware solution in SCIT/HES [3], the architecture can achieve two things: (a) “incomparable” SCIT components to perform intrusion tolerance and (b) increased system dependability.

SCIT has three tunable parameters: the exposure time window W , the cleansing-time $T_{\text{cleansing}}$, and the number N_{total} of total nodes in the cluster. N_{total} , W , and $T_{\text{cleansing}}$ are inter-related. If we want to satisfy a fixed value for exposure window, then a longer cleansing time will require more nodes available for rotation.

Performance. Unlike SITAR and MAFTIA, a SCIT-based system does not require any extra hops or processing of an application transaction. The operations of switching a node from state to state, and of self-cleansing do need extraneous processing and computing resources. But, these SCIT operations happen “out-of-band” and do not interfere with the main data flow of the online node serving incoming traffic. Indeed, testing of SCITized web server showed only a marginal increase in response time under load conditions [4].

Integration. In order to SCIT-ize an application system, application server has to be put under the

control of the Central Controller in charge of the rotation management. Conceptually, this integration is analogous to configuring a node under a system management server in the SNMP world. Thus, from the implementation perspective, no changes are required to the OS, application server, or application software. Integration effort should be light, as compared to involved in the case of MAFTIA.

7. Comparison Summary

Table 1 summarizes the functional differences among the three ITS architectures discussed above, and contains their respective efficiency for intrusion tolerance and survivability.

Table 1. Functional Differences and Efficiency of ITS Architectures.

	SITAR	MAFTIA	SCIT
Function			
Payload Inspection	Yes	No	No
Voting Algorithm	Yes, used to detect faulty replica and survive attacks	Yes, used to detect faulty replica and survive attacks.	No
Deterministic	No	No	Yes
Performance Impact	Impact on response time.	Impact on response time.	Minimal impact on response time. Some impact on computing cycles for starting a new server instance.
Execution of ITS algorithm	In Application Data Flow	In Application Data Flow	Out-of-band
Diversity	Required	Required	Optional, but diversity will make scheme more robust
Recovery	Adaptive recovery performed upon detecting intrusion detection.	Performed upon detecting intrusion. Faulty replica recovered according to healthy ones.	Periodic recovery performed by Controller, based on master copy.
Attack			
CS1 – cross-scripting, SQL Injection	Acceptance Testing	None	None
CS2 – access violation	Access Checker	<ul style="list-style-type: none"> • Authorize Service • Two-level Byzantine agreement protocol • Secret sharing cryptosystem 	None
CS3 – impersonation	None	<ul style="list-style-type: none"> • Java Card protection • Host authorization process 	None
DS – web defacing	Code Test	Prevented by distributed Authorize Service	Limited by automatic recovery.
AS - DoS	Time Test	Distributed IDS Sensors	Limited by automatic cleansing.
ES - data ex-filtration	Reasonable Test.	Authorize Service to some degree.	Short exposure limits ex-filtration.
Protect ITS components	Audit Control and Log analytics	Degree-K resilience.	One-way signal from Controller to servers.
Integration Effort	Configuration of acceptance tests specific to applications.	Applications must use MAFTIA middleware.	Configuration; no change to applications.
Relative Complexity	High	High	Low

8. Conclusion

In this paper, we have studied three typical approaches for intrusion-tolerant system architectures. With the main concepts of extending fault-tolerance mechanisms to the security domain, the approaches essentially differ in how much importance they give to intrusion detection, and how the system responds or eludes cyber attacks. Qualitative analysis was discussed based on case studies of a conceptual web-based OASIS system. Advantages and disadvantages of each approach have been discussed from the perspectives of effectiveness, tunable parameters, performance impact, and integration to application systems.

Some venues can be investigated for future work:

a. Some detection capability can complement SCIT architecture in order to harden the confidentiality protection.

b. Detection-based ITS such as SITAR can take advantage of SCIT self-cleansing approach for its components (proxies, ballot monitors, acceptance monitors) and COTS servers as well. With the application of SCIT to the COTS servers, the probing used to monitor the health of the COTS can be eliminated.

c. ITS architectures can leverage the concept of trust-trustworthiness. For instance, communication between SITAR components can be built on top of a trusted infrastructure like MAFTIA; the Central Controller in SCIT can interface with the cluster nodes via a trusted channel.

The synergetic strengths of the three approaches can be synthesized to design a more comprehensive architecture. Indeed, we are starting to see hybrid architectures such as the “Proactive and Reactive Recovery” scheme in [7].

9. References

- [1] Lala, J. H., Editor, "Organically Assured & Survivable Information Systems (OASIS): Foundations of Intrusion Tolerant Systems," *IEEE Computer Society Press*, <http://computer.org/cspres>, ISBN 0-7695-2057-X2003, 2003.
- [2] Paulo E. Verissimo, Nuno F. Neves, Christian Cachin, Jonathan Poritz, David Powell and Yves Deswarte, Robert Stroud, and Ian Welch. "Intrusion-Tolerant Middleware: The Road to Automatic Security", *IEEE Security & Privacy*, 2006.
- [3] Yih Huang, David Arsenault, and Arun Sood. "Secure, Resilient Computing Clusters: Self-Cleansing Intrusion Tolerance with Hardware Enforced Security (SCIT/HES)", *The Second International Conference on Availability, Reliability, and Security, ARES 2007*.
- [4] Anantha K. Bangalore and Arun K Sood. "Securing Web Servers Using Self Cleansing Intrusion Tolerance (SCIT)", *DEPEND 2009*, Athens, Greece 2009.
- [5] Partha Pal, Franklin Webber, and Richeard Schantz. "The DPASA Survivable JBI – A High-Water Mark in Intrusion-Tolerant Systems", *Workshop on Recent Advances in Intrusion Tolerant Systems '07*, 2007.
- [6] J. Knight, D. Heimbigner. and A. Wolf. "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications", *Intrusion Tolerance System Workshop, Supplemental Volume on 2002 International Conference on Dependable .System and Network*, 2002.
- [7] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, Paulo Verissimo. "Resilient Intrusion Tolerance through Proactive and Reactive Recovery". *13th IEEE International Symposium on Pacific Rim Dependable Computing*, 2007.
- [8] Q. Nguyen and A. Sood. "Quantitative Approach to Tuning of a Time-Based Intrusion-Tolerant System Architecture". *3rd Workshop on Recent Advances on Intrusion-Tolerant Systems*, Lisbon, Portugal, 2009.
- [9] Alfonso Valdes et al. "An Architecture for an Adaptive Intrusion-Tolerant Server". *LNCS Springer/Berlin, Volume 2845/2003*, pp. 569-574.
- [10] James C. Reynolds et al. "On-Line Intrusion Detection and Attack Prevention Using Diversity, Generate-and-Test, and Generalization". *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003.
- [11] Dick O'Brien et al. "Intrusion Tolerance Via Network Layer Controls". *Proceedings DISCEX 2003*.
- [12] Partha Pal et al. "An architecture for adaptive intrusion-tolerant applications". *Softw. Pract. Exper. 2006*; 36:1331-1354.
- [13] Maha Sliti et al. "Intrusion-Tolerant Framework for Heterogeneous Wireless Sensor Networks". *IEEE AICCSA 2009*, pp. 633-636.
- [14] Tao Zhang et al. "Building Intrusion-Tolerant Secure Software". *IEEE CGO 2005*.
- [15] Khin Mi Mi Aung, Kiejin Park and Jong Sou Park. "A Rejuvenation Methodology of Cluster Recovery". *CCGrid 2005, IEEE International Symposium Vol. 1*, pp. 90 - 95, May 2005.
- [16] Paulo Sousa et al. "The FOREVER Service for Fault/Intrusion Removal". *WRAITS 2008, Glasgow, Scotland*.
- [17] Hans P. Reiser et al. "Hypervisor-Based Efficient Proactive Recovery". *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, 2007.