AFRL-IF-RS-TR-2004-206
Final Technical Report
July 2004

ORGANICALLY ASSURED AND SURVIVABLE INFORMATION SYSTEMS (OASIS) DEMONSTRATION AND VALIDATION PROGRAM

The Boeing Company Boeing Phantom Works

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. N953

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-206 has been reviewed and is approved for publication.

APPROVED: /s/

PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JULY 2004	**3. REPORT TYPE AND DATES COVERED** Final Aug 02 – Oct 03

4. TITLE AND SUBTITLE
ORGANICALLY ASSURED AND SURVIVABLE INFORMATION SYSTEMS (OASIS) DEMONSTRATION AND VALIDATION PROGRAM

5. FUNDING NUMBERS
C - F30602-02-C-0203
PE - 63760E
PR - N953
TA - OA
WU - IS

6. AUTHOR(S)
Daniel Scnackenberg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Boeing Company Boeing Phantom Works
Engineering and Information Technology
PO Box 3999
Seattle Washington 98124-2499

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-206

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Patrick M. Hurley.IFGA/(315) 330-3624/ Patrick.Hurley@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

This document summarizes the results of the Boeing team's survivable JBI design effort. The objective of the effort was to develop the design for a JBI system that can operate through sophisticated adversary attack, and yet scale well to wide-area networks. The system design approach ensures that the intrusion tolerance mechanisms do not significantly degrade performance, while still tolerating compromised components. For the system to be deployable and the results to transition to real systems, the performance cost of providing intrusion tolerance must be contained. The design trades opted for solutions that preserve system performance. The final report includes a section on the system design, a summary of system requirements and a concept of operations, design details and a summary of project accomplishments, planning documentation, lessons learned and a conclusion.

14. SUBJECT TERMS
Orgainically Assured And Survivable Information Systems, Intrusion Tolerant Survivable System, Application-Level Authorizations, Enclave-Level Correlation, Intrusion Reporting, And Intrusion Response, Dynamic Behavior, Hybrid To Byzantine Fault-Tolerant Approach

15. NUMBER OF PAGES
32

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1.0 Introduction

This document summarizes the results of the Boeing team’s survivable JBI design effort. Our objective in this effort was to develop the design for a JBI system that can operate through sophisticated adversary attack, and yet scale well to wide-area networks. The system design approach ensures that the intrusion tolerance mechanisms do not significantly degrade performance, while still tolerating compromised components.

For the system to be deployable and our results to transition to real systems, the performance cost of providing intrusion tolerance must be contained. Our design trades opted for solutions that preserve system performance.

Section 2.0 provides an overview of the system design. Additional design information can be found in project documents. [1] provides a summary of the system requirements and [2] provides the concept of operations. [5] and [7] provide design details. Section 3.0 summarizes the project accomplishments. One of the key accomplishments is the development of an assurance argument for the survivable JBI ([3] and [4]). Related to the assurance argument is our coverage of the attack space with our defense and detection mechanisms. This coverage is presented in [10]. [6], [8], and [9] provide planning documentation for Phase 2.

Section 4.0 provides conclusions from our effort, including lessons-learned during the design of a survivable JBI.

2.0 Design Overview

Our survivable JBI system design supports JBI operation over both local and wide-area networks as illustrated in Figure 1. This figure shows multiple survivable JBI enclaves, each comprising a data and management LAN. The data LANs in remote enclaves communicate over a data IPSec VPN, and the management LANs communicate over a separate VPN. These VPNs separate data and management traffic, reducing the risk that penetration of the data LAN will lead to penetration of the management LAN.

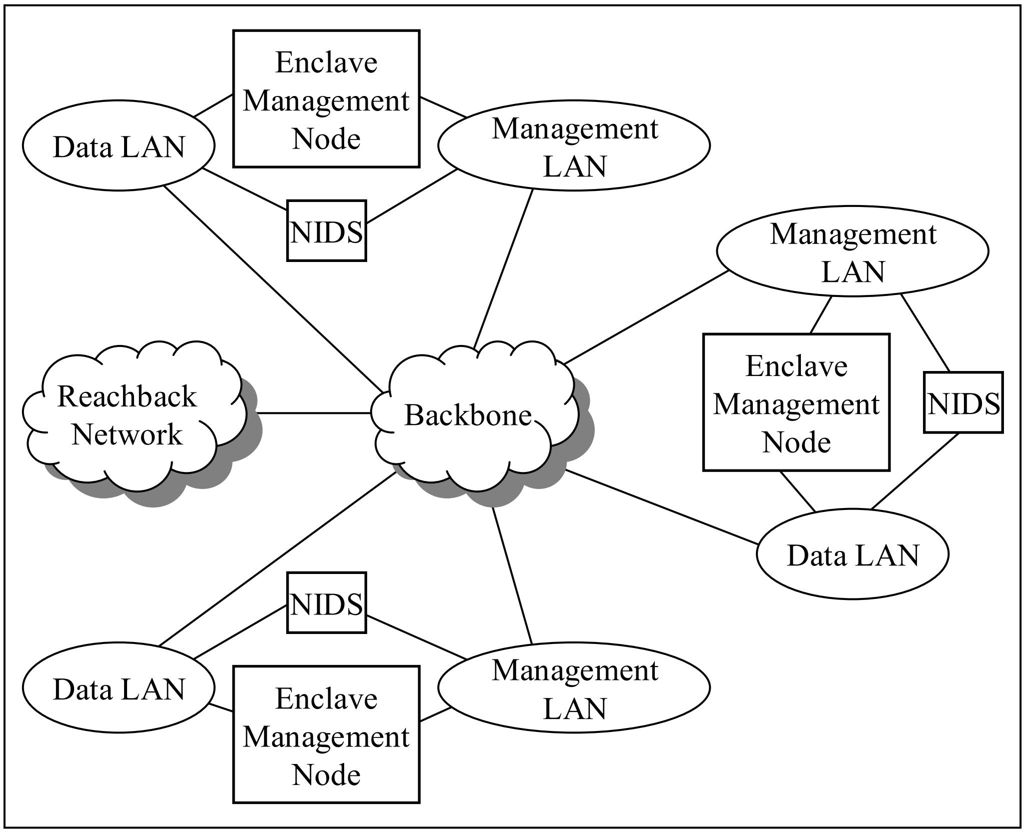

Figure 1. Survivable JBI Across Wide-Area Networks

The data and management LANs are connected only through an enclave management node. The enclave management node permits management control information to flow from the management LAN to the data LAN, and intrusion alerts to flow from the data LAN to the management LAN. A network intrusion detection system (NIDS) also has interfaces on both LANs; however, on the data LAN, the NIDS is configured for only promiscuous monitoring, so the NIDS cannot send packets to the data LAN.

Figure 2 shows the configuration of the management LAN. Each management LAN has the capability to configure any component in the system. The Embedded Firewall (EFW) policy server configures EFWs throughout the system. The management control node uses a technology called survivability grammar to configure the distributed algorithms in the system and hosts the system-level correlation, intrusion reporting, and intrusion response mechanisms.

The enclave management node provides a bridge between the management and data LANs, as well as enclave-level correlation, intrusion reporting, and intrusion response.

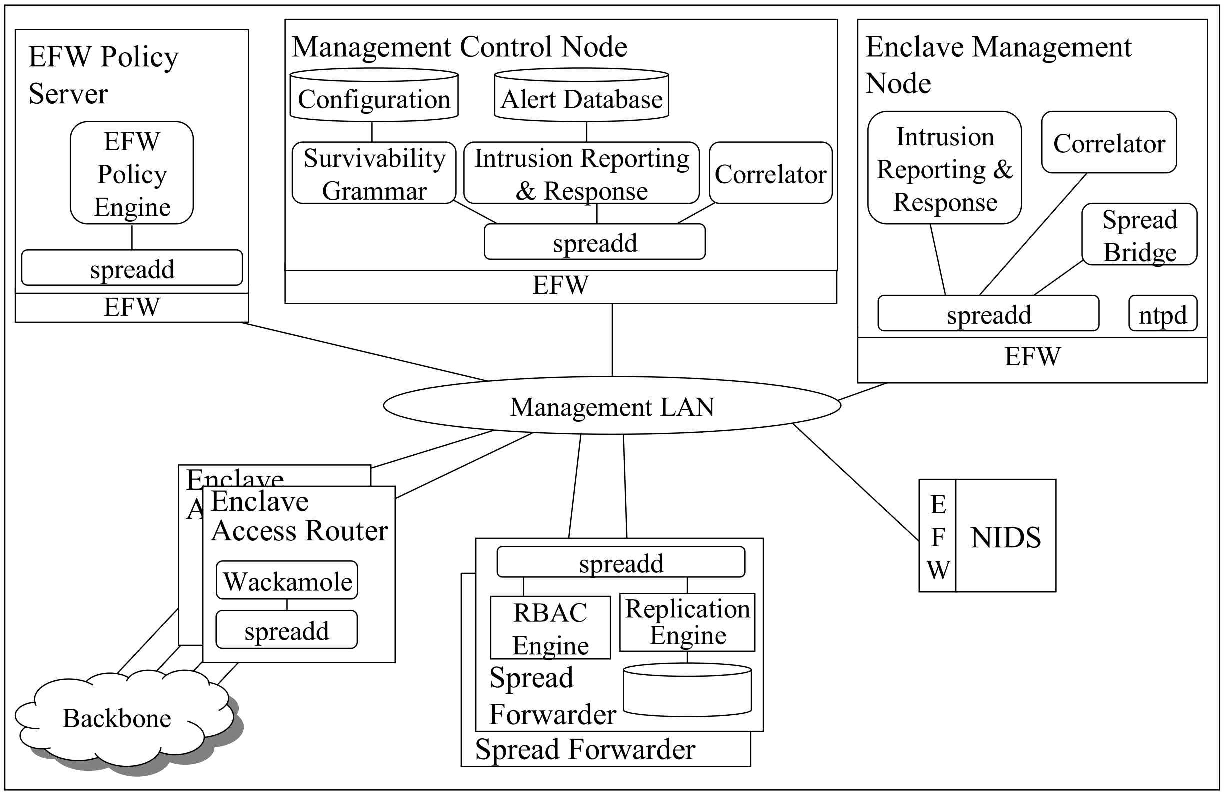

Figure 2. Management LAN Components

A key element in our design strategy is to provide survivable JBI services across wide-area networks without incurring significant performance impacts. To achieve this capability, we use a hybrid approach for intrusion tolerance that leverages a trusted component – the Spread forwarder – to provide intrusion tolerance without the performance overhead seen with Byzantine fault-tolerant algorithms. Figure 3 shows the components of the data LAN. Key aspects of the data LAN design are described in section 2.2. Our design implements eight layers of defense, each responsible for participating in covering some of the system survivability objectives and threats. Separate documents provide our detailed design and assurance arguments, which show how to integrate these layers systematically to satisfy end-to-end JBI survivability requirements. The assurance arguments iteratively refine JBI survivability requirements into those at lower layers until only system assumptions remain.

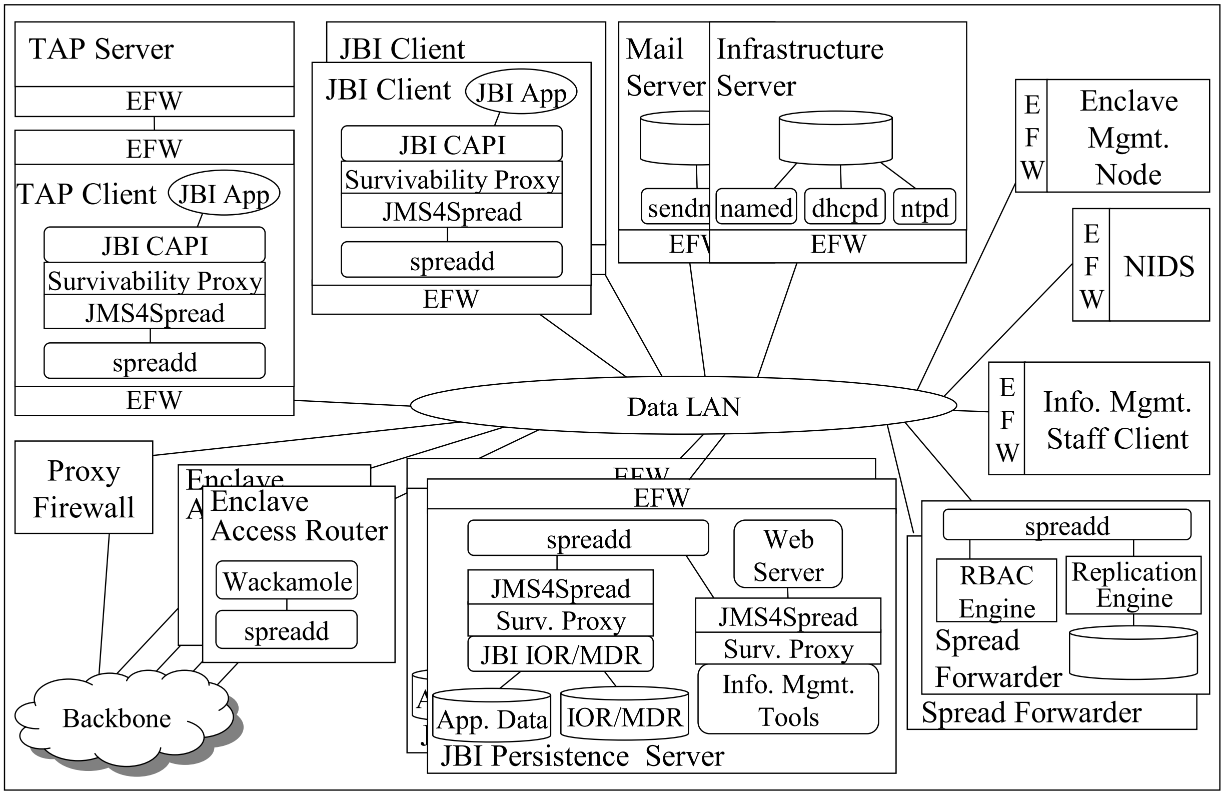

Figure 3. Data LAN Components

2.1 Survivability Objectives

Our design achieves the following objectives:

1. Data Integrity. Essential and mission-critical data cannot be altered by an adversary.
2. Data Confidentiality. Essential and mission-critical data cannot be read by the adversary.
3. Availability. Mission-critical operations complete within the allotted timeframe and all functions, except non-critical functions, are available for use at all times.

2.2 Defense Layers

Our project implements eight layers of defense, each responsible for participating in covering some or all of the above objectives.

1. Practical network topology and design. The network provides pervasive deployment of packet filtering and rate limiting to defend against both internal and external attacks. All packets are accurately classified as data traffic, management traffic, or traffic to the general Internet. Appropriate routing, rate limiting, and packet filtering rules are defined and enforced for each traffic category. Within the management and data categories, fine-grained enforcement is performed based on accurate source address information for each packet. Hardware defenses are used against layer-2 attacks such as ARP-cache poisoning or deceptive use of link addresses on multicast packets. Network liveness is maintained through secure router configuration, a multiply-connected physical network topology,

and network-wide bandwidth management. Network intrusion detection technology is deployed pervasively and is provided with network support including robust and secure communication channels for data aggregation and hardware suppression of address spoofing for packets that originate locally.

2. Trusted key hardware. Our design relies on two components that are virtually impenetrable: the Boeing Secure Network Server (SNS) and Embedded Firewall (EFW). Using a component that is impenetrable simplifies the design for critical functions, providing a simple approach to tolerating Byzantine behavior elsewhere in the system without significant performance costs usually associated with symmetric solutions. The SNS supports both Survivable Spread and JBI database replication. The EFW provides pervasive packet filtering, which provides strong containment of compromised devices. These devices cannot change their packet filtering rules, and hence are severely limited in the further damage they can cause.

3. Hardened operating system environment. Our design leverages a number of technologies to harden platforms and applications against attack and to tolerate compromises by containing the effects of an attack. RaceGuard, PointGuard, and FormatGuard prevent many exploits from achieving their objectives. RaceGuard kills the attacking process, while PointGuard and FormatGuard kill the vulnerable process under attack turning the attacker's penetration attempt into a crashed process. Domain type enforcement (DTE), wrappers, CryptoMark, and Guaranteed Internet Stack Utilization (GINSU) provide operating-system-level containment of applications and thereby enforce limits on the resource usage and access of every application or service. Thus, compromised applications or services cannot impact the mission. The use of strict DTE and wrapper policies also greatly increase our ability to detect and immediately terminate malicious processes. Beyond these capabilities, we use both general host-based intrusion detection and custom detectors to provide a high likelihood of attacker detection either on the attacker's use of an exploit or the subsequent actions of a penetrated process. We categorize our intrusion alerts into high and low confidence, and associate severe reactions, such as process kill, with only the high confidence rules. The system only issues alerts for the low confidence rules to prevent denial of service (DoS) inflicted by way of the intrusion detection and response system.

4. Intrusion tolerant survivable message bus. Our design leverages the robust network and trusted hardware to provide a survivable version of the Secure Spread toolkit, called Survivable Spread, which provides group communication, publish-subscribe, and message bus services, while tolerating Byzantine failures. Survivable Spread leverages EFW-enforced use of IP multicast via Ethernet multicast or broadcast within an enclave. It places all the inter-enclave forwarding functionality on a small number of daemons running on the Boeing Secure Network Server hardware. Each enclave contains two trusted forwarders improving availability. Given that the Secure Network Servers are impenetrable, trusted forwarders on the Secure Network Servers are not compromised. This design enables tolerance of compromised Spread daemons by having the forwarding node eject misbehaving nodes. Our solution enables us to leverage Spread's scalability and still achieve high assurance.

5. Active replication through global persistent consistent order. Use of a verified replication engine algorithm operating on the trusted SNS enables efficient, reliable replication of

the JBI information object and metadata repositories (IOR and MDR) in the face of node crashes and network partitions. This service also supports efficient recovery of failed JBI persistence servers. This SNS-resident *replication engine* leverages Survivable Spread's delivery guarantees to achieve database consistency. After ordering database related messages, the replication engine signs each message and sends it to the persistence server, which updates or queries the database.

6. Byzantine fault tolerant application data access. To tolerate Byzantine failure in our persistence servers, our design uses an algorithm that requires *f*+1 replicas to tolerate *f* failures. Clients perform IOR and MDR database queries and must receive responses from at least *f*+1 servers. If the client receives *f*+1 consistent responses, then the data is correct. If not, the client can determine which of the *f*+1 are correct. The servers store the replication-engine-signed entries, which prevents Byzantine servers from generating new database entries without detection. The replication server also stores database delete messages. When a client detects that some responses differ, that client queries servers that differ for a history of database updates (inserts and deletes) and uses this information to determine which servers are lying.
7. Support for JBI client recovery. Our design uses passive replication techniques applied to the user-oriented and non-deterministic applications that comprise many of the JBI legacy and client applications. Applications can fail-over and restart on new replicas using JBI application state data preserved through our persistence service. Heterogeneous replication of the application platforms limits common mode failures to reduce the likelihood of repeated application failures. These mechanisms enable application reconstitution on a backup platform. Other applications may also benefit from this mechanism; however, to support recovery the applications must save sufficient application state.
8. Application-level and group communication level access control. Our design uses strong authentication mechanisms to support application-level authorization to control access to JBI application services. JBI application authentication and authorization is accomplished through Survivable Spread mechanisms.

Our design provides intrusion tolerance without sacrificing scalability and performance.

2.3 Addressing Threats

The following table shows how the different layers described above support the survivability objectives.

Survivability Objectives	Supported by Defense Layers
1 – Data Integrity in transit	1, 2, 4
1 – Data Integrity in memory (volatile)	3, 8
1 – Data integrity in storage (persistent)	3, 5, 6, 8
2 – Data confidentiality in transit	1, 2, 4
2 – Data confidentiality in memory	3, 6, 8
2 – Data confidentiality in storage	5, 7
3 – Applications execute on time	3, 5, 6, 7
3 – Messages delivered on time	1, 2, 4

Figure 4. Layers of Defense

The defense layers provide protection against basic system threats of benign network and computer faults, passive attacks, low intensity active attacks, denial of service, and internal attacks. The following paragraphs describe briefly the protections for each of these threats.

2.3.1 Benign network and computer faults

Our network topology provides redundant connectivity between enclaves. Survivable Spread is able to detect such failures and provide a strong semantics service that enables building global consistent persistent order. This, in turn, enables consistent replication of critical data and services in spite of network component failures and recovery, and network partitions and merges.

2.3.2 Passive attacks

On the network level, all non-reachback wide-area communication is encrypted using IPSec. For all mission-critical and management data, Survivable Spread encrypts all daemon level communication. Group keys at the Survivable Spread client level are used to encrypt application data. The group key distribution mechanisms ensure that group keys exist only on machines that have Spread clients belonging to the group. Note that to belong to the group, and hence acquire the group key, the client must be authorized to belong to the group. Thus, Spread client traffic is protected end-to-end on the wire and in intermediate devices; Spread control traffic is protected between Spread daemons; and non-Spread traffic (as well as Spread traffic) is protected on the wire between enclaves.

2.3.3 Low intensity active attacks

Pervasive packet filtering is provided at the network level and by the EFW trusted hardware. This filtering blocks most active attacks from entering the network, or on entering the network from reaching the intended victim. Enclave proxy firewalls block attacks along the highest risk attack path – the path to the reachback network. Network monitoring by the network intrusion detection system (NIDS) provides another layer of defense, detecting both violations of the packet filtering rules and common network exploits. Our hardened platform capabilities block most known and likely penetration attacks, and contain the remaining successful penetrations.

Failure detection and recovery mechanisms enable restart on low intensity DoS attacks. Survivable Spread cryptographic mechanisms prevent packet insertion and replay attacks at that level. Intrusion detection and response provides local reaction to contain many intrusions, and global reporting to enable operator situational awareness.

2.3.4 Denial of service

Redundant network design provides alternative paths. Backbone and enclave access routers and the reachback packet filter all rate limit traffic to ensure that each class of traffic has sufficient bandwidth. Trusted EFW hardware on vulnerable hosts also rate limit that host's traffic. Similar rate limiting is done in each host operating system. Within hosts, GINSU limits consumption of network-related resources at the application granularity to ensure that one application cannot deny service to another within the host. Survivable Spread is able to work through denial of service attack by adaptively employing more aggressive control messages as observed loss rate is increased. This enables the mission-critical and management traffic sent over Spread to be delivered in spite of attack. The NIDS provides redundant detection of rate monitoring failure in case other preventative mechanisms are subverted.

2.3.5 Internal attack

The combination of the trusted EFW and the platform hardening mechanisms contain the actions of malicious processes both within its platform and on the network, greatly constraining what else these processes can attack. The NIDS provides detection for those network attacks that evade the preventative mechanisms.

Use of enforced IP and Ethernet multicast for all Spread traffic, and use of trusted nodes (the SNS) provide protection from multi-faced behavior (where a participant sends contradicting messages to different parties) and protocol disruptions at the messaging level. The penetration resistance of the SNS enables it to act as a trusted point for operations such as expulsion of misbehaving Spread daemons and forwarding of Spread messages to other enclaves, as it will not have two-faced behavior or attempt to corrupt the protocol. This achieves tolerance of Byzantine Spread daemons without the high cost of Byzantine agreement algorithms. An alternate, simpler setup of Survivable Spread configures a single daemon on the trusted forwarder. All clients communicate with this daemon through authenticated secured point-to-point channels. This solution avoids all multi-faced daemon behavior and is simpler and therefore higher assurance. Naturally, it is considerably less scalable as its performance decreases linearly with the number of group members per LAN. Nevertheless, our design allows switching between these setups on the fly (a capability already available in Spread).

Above the Survivable Spread level, our design provides active replication of the JBI persistence server; again leveraging the trusted node we have in each enclave – the SNS – and a verified replication engine algorithm that leverages Spread's delivery semantics. This design tolerates Byzantine failure of JBI persistence servers through an algorithm that requires only f+1 replicas to tolerate f failures. This algorithm leverages specific data access semantics used by the JBI (i.e., no aggregation operations), which allows a significant reduction in the number of replicas required at the cost of some additional database storage.

2.4 Deception

A major objective of deception is to make it much more difficult for the adversary to deliver to their target the packets necessary to compromise an application or service. Several deception techniques are available, although some do not provide the desired effects in our proposed configuration. The following list discusses types of deception and their role in our design.

- Honeypots. Honeypots have limited value when the attacker knows a honeypot is likely. Most honeypots do not fool a knowledgeable adversary for long, but it may distract the adversary for a short time. In the planned red team experiment it is unlikely that today's honeypots would provide a significant barrier to the red team; however, adding a honeypot into the design is relatively straight-forward, so that one could be added for the sake of the experiment without significant changes to the architecture.

- Platform/application disguising. A number of technologies (e.g., U. Mich. Protocol Scrubbers) exist to remove platform or application signatures from network traffic. These technologies are particularly useful when the adversary has little advance knowledge of the system configuration because they are likely to have to try multiple exploits to get the right one to the right device. This capability would be straightforward to add to our design by incorporating this feature in the reachback packet filter. Elsewhere, information used by adversaries is hidden from devices outside an enclave through encryption. The reachback packet filter would only obscure the reachback traffic, which is served by a Sidewinder firewall. These have been shown to be relatively impenetrable, so the benefit for this design is low.

- IP Address hiding. One technique for address hiding within a system is DyNAT. DyNAT uses continuously changing addresses between DyNAT devices, which makes it difficult to determine which device is a server and which a client. It is also difficult to target a device because the adversary does not know what address will be used next unless the adversary is already on a machine that is participating in the DyNAT system. The problem with DyNAT is that if one device in the DyNAT system also connects outside the DyNAT system then that device's vulnerabilities become the system's vulnerabilities. An attacker entering through that path would become an insider, and hence could send packets to the real target machines. Thus, DyNAT is suitable only for closed systems or it requires that only impenetrable devices communicate outside the closed system. Because our design attempts to simulate an open system, DyNAT is not suitable.

  We get many of the benefits of address hiding from our pervasive packet filtering and DTE mechanisms. An adversary is severely limited at multiple points as to which packets can be targeted at any specific machine. Further, if the adversary gains control of a process on a machine, DTE and the EFW constrain what packets the adversary can cause to be emitted by the node.

- Traffic hiding. In our design, Spread can use deception to defend against attacks where the adversary has direct access to a LAN segment. All messages that Spread sends on the LAN are multicast, which makes it difficult without prior knowledge to determine the identity of the recipient and hence the role played by each node. To prevent the attacker from effectively analyzing traffic patterns, Survivable Spread will include a mechanism to insert additional random traffic from various nodes to obscure component roles in the system. Spread's cryptographic mechanisms hide the data, which would be the other clue of component roles.

- Code address hiding. Random pointers used by PointGuard provide another form of deception. An adversary does not know the specific packet needed to penetrate a device because it is likely that a buffer overflow attack will change a pointer value that causes a segmentation fault, thus transforming a potential takeover into a crash.

2.5 Dynamic Behavior

Our design has several dynamic reactions available to be used when network security state changes. These dynamic behaviors generally reduce performance or functionality to improve system intrusion tolerance.

- Disable non-critical services at the host, EFW, enclave access router, or reachback router.
- Reduce backbone bandwidth allocated to non-critical services.
- Change GINSU policy to reduce non-critical application resources.
- Dynamically adjust to changes in Spread token loss rate to create a token that is sufficiently robust given current traffic volumes.
- Adjust the amount of traffic used for Spread for deception.
- Global change to one of a predefined set of policies by all components within an enclave or the system.
- Restarting/relocating crashed applications.
- Killing and restarting compromised applications
- Rebooting compromised or crashed machines.

These reactions are invoked based on the detection of behaviors that violate policy or are known attacks. To prevent attackers from using the reactions against the system, reactions that have the potential for interfering with the mission, such as restarting a critical computer, are only taken in response to alerts that unequivocally indicate that a compromise has occurred or is occurring. Alerts for which detection is less certain are reported to an operator, who can judge how to handle uncertain detections. Thus, unnecessary self-inflicted DoS that impacts mission operation is avoided.

2.6 High Detection Rates

The combined use of standard intrusion detection mechanisms for both network- and host-based detection, coupled with the use of access control mechanisms and other custom detectors to trigger alerts when applications attempt to step outside their tightly defined environments, enables detection of both the attacker's exploit, and if successful, the attacker's subsequent actions. Use of access control mechanisms that are tailored to provide minimal application access to resources (e.g., files or network resources) was shown in the IA program's Integrated Feasibility Experiment (IFE) 2.3 to be extremely effective at detecting attacker behavior once the attacker has gained access to the local machine. This works well in highly constrained application environments as we have in our design.

Alert correlation mechanisms at the enclave and system level allow operators to significantly reduce the number of low-level alerts displayed by viewing only correlated alerts.

3.0 Project Accomplishments

Our design effort achieved the following contributions–

- Use of an assurance argument during the design process to provide guarantees of system-level properties
- Use of a hybrid approach to Byzantine fault-tolerance, leveraging a small number of trusted components to gain intrusion tolerance without significant performance impact
  - Survivable group communication
  - Intrusion-tolerant database replication - unique algorithm requiring only 1 correct replica compared with standard algorithms requiring ⅔ of the replicas to be correct
  - Consistent deployment of system access control policy to distributed replicas
  - Client checkpointing and recovery
- Use of varied intrusion tolerance approaches
- Pervasive use of a single group communication paradigm to achieve
  - Publish-subscribe message delivery services
  - Replicated data storage
  - Failover
- Pervasive packet filtering and strong access control to narrow adversary attack vectors and contain attack
- Use of logic-based survivability grammar to drive system configuration

The following sections describe our approach in these areas.

3.1 Assurance Argument

The ideal assurance argument would provide a formal proof that the system achieved all of its security claims, and denied the attacker from achieving any objectives that were counter to those claims. Such a proof would rely on exhaustively showing how the architecture thwarted all possible ways for the attacker to achieve the objectives. Therefore, a traditional requirements-driven software engineering approach would require that the system designers first identify all of the potential attacker objectives and the ways that the attacker could achieve those objectives. If the architecture blocks all of the attack paths, then the system is secure. This approach may be complete, but we encountered many difficulties when trying to implement it.

One of our early approaches was an attempt to start from a *threat analysis* and identify all of the possible ways that an attacker could realize those threats using *attack trees.* The threat analysis attempted to identify all of the potential attacker objectives with respect to violating the availability, integrity, or confidentiality of the system. The attack trees then attempted to identify the exact means that might be available to the attacker to achieve these objectives. Our methodology was as follows:

- Identify the attacker starting point as a set of resources (possibly just external network access).
- Enumerate the resources that the attacker must acquire to achieve the objective.
- Identify all of the possible ways of acquiring each of those resources, including both technical and non-technical means.
- Iteratively repeat the analysis, identifying intermediate resources required to obtain the desired resources that result in achieving the objective. Stop when only the resources available at the starting point are necessary.

The analysis resulted in a directed graph showing all of the potential sequences of steps that an attacker could take to achieve an objective (realize a threat). The nodes of the graph are the required resources and the edges are the means of obtaining those resources. If some element of the security architecture counters at least one edge in every path in the graph from the attacker starting points to the objectives, then the system is secure.

Although falling short of a formal proof, this method would have provided some comfort as to the completeness of the assurance argument. It exhaustively examined the attack paths using a "divide and conquer" strategy that had a high likelihood of being complete for any particular node in the attack path. The disadvantage of this method is that attack trees do not scale well for large systems. We made several attempts to improve attack tree scalability, with some success. However, even with those improvements, the method remained extremely labor intensive and time consuming. After many person-months of effort, we halted the attack tree effort in favor of *claim trees*.

Claim trees show how the system achieves the desired high-level assurance properties claimed for the system. The argument for each high-level objective (claim) has a rationale and list of dependencies and assumptions that are part of the rationale. Those dependencies become lower level objectives that have further rationale, assumptions, and dependencies. The strength of claim trees is that the authors start with a limited set of desirable properties and can form a coherent argument as to how the system provides those properties. The disadvantage is that it is very difficult to prove that all dependencies and assumptions have been identified. If dependencies and assumptions are omitted, then there could be undocumented ways to attack the system through those unidentified dependencies.

The claim tree and threat analysis/attack tree methodologies are complementary and approach the same problem from different perspectives. This is not a new concept; the NRL version of claim trees includes a notation to describe which threats are blocked by an aspect of a given objective. Attack trees help identify the assumptions and dependencies in the claim trees. For example, the threat analysis would indicate one possible attacker objective as:

> "Delay processing of air tasking orders by flooding the local area network, preventing communication between JBI clients and servers."

This is a threat against the system liveness claim in the claim trees. The claim tree should show that system liveness is dependent on network liveness. The attack trees would show all the ways that the attacker could flood the network to block transmission of legitimate packets. This list of possible attack paths should help identify the dependencies of the network liveness claim. The rationale for the network liveness claim should state how each of the attack paths is blocked. Given that this is a relatively high-level claim, the rationale has several dependencies on other claims in order to block the attack paths.

Given more time and resources, we may have chosen an approach based on both the claim trees and the attack tree analysis. We chose to focus our efforts on the claim trees because it was achievable in the time available. We endeavored to apply some of the attack tree thought process in our claim tree development by examining each claim and trying to identify the possible ways to defeat the claim. This mental process helped reveal missing dependencies and improved the completeness of our argument.

This process led us to conclude that a thorough assurance argument must analyze the system from the perspectives of both the both defender (claim tree approach) and the attacker (attack tree approach). The claim tree approach forces the analyst to make sure that all of the countermeasures actually contribute to the goal of achieving the high-level objectives. The attack tree approach reminds us to think of all of the unintended ways that the system may be used and helps provide completeness in the assurance argument.

3.2 Hybrid Approach

Through our design process, we arrived at an architecture that leverages a few trusted components to achieve Byzantine fault-tolerance with minimal performance degradation. This approach deviates from traditional symmetric Byzantine fault-tolerance algorithms, where each component has an equal role, by designating specific components as not susceptible to Byzantine behavior. With these trusted components, our design is able to eliminate the possibility of two-faced behavior by Byzantine components without the high cost of standard algorithms.

This approach was taken primarily to preserve scalability across wide-area networks, where the large number of messages required for standard Byzantine algorithms makes their use impractical. Our approach is similar to that described by Paulo Verissimo [11] as the hybrid approach. Verissimo advocates this approach because it can achieve high intrusion tolerance and still provide good performance.

In our design, we leverage one trusted component to provide intrusion tolerance – the Boeing Secure Network Server (SNS). The SNS provides a high assurance computing platform, which has been formally specified and verified at the design level. This component is used to implement selected algorithms necessary to ensure survivable group communication, database replication, access control policy distribution, and client checkpointing and recovery.

Throughout our design, we use a common group communication paradigm based on Secure Spread to provide communication support for JBI functionality, as well as for monitoring and control. To support survivable group communication, the SNS implements the Survivable Spread group communication protocol and acts as the forwarding node between enclaves. The Survivable Spread implementation on the SNS can detect malicious behavior on the part of other Spread daemons, and removes misbehaving daemons from the Spread network. The algorithms for detecting misbehaving daemons are described in more detail in [7] and [4]. The use of the SNS, coupled with guaranteed multicast of messages within a LAN segment by all Spread daemons ensures that no daemon can exhibit two-faced behavior or violate the Spread protocol.

Survivable database replication is required to support persistence of JBI information objects. Our design has the following primary database replication requirements: (1) consistent database content across replicas, (2) tolerance of network partitions and joins, and (3) tolerance of Byzantine behavior by some database replicas. Our design uses the SNS to ensure these characteristics. The SNS executes the replication algorithm described in [12] to ensure that database replicas are consistent and tolerate network partitions and joins. Once a total order has been determined for database operations, the operations are time-stamped and signed by the SNS and sent to the database replicas. Because the SNS is trusted to exhibit only non-Byzantine behavior, the approach ensures that correct database replicas receive and execute database operations in the same order, ensuring consistent access by clients. Because the operations are signed by the SNS, they can be validated by database clients, enabling clients to detect malicious database replicas. Because of the nature of database access in the JBI, clients can determine which database replica responses are correct given that at least one correct replica response is received.

Access control policy distribution is also accomplished using the replication engine on the SNS. The requirement is that access control policy be consistently enforced across database replicas, and thus the access control policy changes must be consistently ordered with the database operations across replicas. Using the same replication engine mechanisms ensures this ordering.

Another use of the replication engine on the SNS is to support client checkpointing and recovery. Client checkpoint data is distributed across the system using the replication engine on the SNS to ensure consistent replication of this data across all correct database replicas. This mechanism enables client recovery anywhere in the system based on the last client checkpoint.

3.3 Varied Intrusion Tolerance Approaches

During the design of the Survivable JBI, we concluded that a single approach to intrusion tolerance was not appropriate for all parts of the system. Specifically, in some cases, resource utilization determined which intrusion tolerance approach to use, and in other cases technical limitations were the determining factor. Because of varying requirements and constraints on the multiple system components, our design incorporates several strategies to provide intrusion tolerance for those components.

The generally accepted strategy for building an intrusion tolerant service from a non-intrusion tolerant service on an untrusted platform is to replicate the service and use Byzantine agreement to maintain consistency between the service replicas. The benefits of this approach are maintained availability and integrity of the service in the event of replica compromise, and its support for an arbitrary service interface. We used this approach to provide intrusion tolerance for the persistence service since the persistence service is a core component upon which other components rely. To do so we replaced the multi-threaded JBoss server with a simplified persistence server that runs the same code, but eliminates the non-determinacy issues related to having a multi-threaded server.

The drawbacks to an actively replicated service with Byzantine agreement are–

- High resource utilization to have multiple replicas.
- Reduced performance due to Byzantine agreement protocols.
- The constraint that non-deterministic applications and applications with user interfaces are difficult to replicate actively.

For the persistence service, the increased resource utilization was justified by the need for high availability and integrity of that core component. The performance issues were addressed, as discussed in the previous section, by using a few trusted components that eliminated the need for expensive Byzantine agreement protocols. Finally, the non-determinism issues in the persistence service were easily eliminated by making the service a single-threaded deterministic application.

However, a different solution was needed for the JBI clients because the resource cost of actively replicated clients could not be justified, and several technical constraints made it difficult to replicate the JBI clients actively. These constraints included large legacy code bases with non-deterministic behavior and user interfaces that are not usable when actively replicated. Therefore, instead of using an active replication approach for the JBI clients we used an approach that combined passive replication with limited checkpointing to provide high availability.

The disadvantage to that approach is that if an attacker could successfully compromise a client without being detected, then there would be no means to stop the attacker from accessing the system. However, a successful undetected compromise of a JBI client is a highly improbable event given the countermeasures that we have put in place on the node. Such countermeasures include an embedded firewall, DTE, StackGuard and FormatGuard, and other technologies that convert potential Byzantine failures into crash failures.

There are several advantages to the passive replication with limited checkpointing approach. First, the resource requirements are low because each client initially needs only a single node to run on. A small number of backup nodes can be shared by all of the clients and used in the event that a client needs to be restarted on a new node. Second, the performance impact is marginal since no replica consistency is required. The only additional processing required by the client is to store the information objects that it receives and publishes (or references to those objects) with the persistence service where they can be recovered by a recovering instance of the client. Finally, this approach works for clients that have non-deterministic behavior and user interfaces because only one version of the client is running at a time.

By using several different intrusion tolerance approaches for different components in the system, we were able to maximize the benefit/cost ratio for each system component and the system as a whole. These design choices allowed maximum flexibility in the design to incur costs where additional protection was needed, and to avoid costs where less expensive intrusion tolerance techniques were sufficient.

3.4 Group Communications

Our design leverages the Spread group communications system to support most of the communication requirements within our Survivable JBI system. By using a single group communication paradigm, we achieve consistent behavior across all of the distributed components in our design. The use of a single communications package reduces the risk of residual vulnerabilities: we only need to secure one code base. Spread provides scalable communication across both local and wide-area networks. Spread supports extended virtual synchrony delivery semantics, which enables development of distributed algorithms on top of Spread that tolerate network partitions and joins.

Our design leverages Spread's properties to provide the following services–

- JBI publish-subscribe services. JBI publish-subscribe support is provided through Spread's multicast messaging service. This approach removes a single-point of failure that exists in publish-subscribe architectures that require a central server to support publish-subscribe functions. Each publisher sends messages to a Spread group, and Spread uses its reliable multicast service to deliver these messages to subscribers for that group that are in the current view of the publisher. Note that subscribers that are not currently reachable by the publisher can retrieve the message later from the persistence server once connectivity is restored.

- JBI information object persistence in multiple database replicas. The JBI CAPI specifies the capability to make objects persistent, enabling clients to query the system for those objects later. Our approach uses a replication engine on the SNS to order these database operations correctly for multiple database replicas. This replication engine uses Spread's extended virtual synchrony delivery semantics to provide correct replication in spite of network partitions and joins.
- Failover support. The failover mechanisms used in the Wackamole router, the transmission of data between management and data networks, and the client checkpointing and recovery all use Spread's extended virtual synchrony semantics to ensure consistency between replicas.
- Monitoring and control. Our design uses Spread's reliable delivery semantics to transmit monitoring data (e.g., intrusion alerts) to management components. Control messages from management components to other components using Spread.

3.5 Defense Mechanisms

Although our design uses several defense mechanisms to achieve system survivability, two mechanisms play a key role: pervasive packet filtering and strong access control.

Our design uses packet filtering in multiple locations to restrict the packets an adversary can successfully send to a system component. Packet filtering is performed at the reachback network boundary, within the inter-enclave network backbone, at the enclave access routers, and in each device within the data and management LANs. Packet filtering on devices within the data and management LANs occurs both at the operating system level and within embedded firewalls (EFW), which are packet filtering network interface cards. These EFWs are controlled by a central management component and cannot have filtering rules altered by the host. Thus, if an adversary compromises EFW-protected components, the adversary still is constrained by the EFW packet filtering rules. This severely limits what the adversary can do to the rest of the system from a compromised component. Even if there are vulnerabilities that could be exploited on another component, if the adversary is restricted from accessing the vulnerabilities, further attack progress is limited. Packet filtering within the rest of the network limits attack from outside to only those few services made accessible.

The second mechanism that is used throughout the system is domain-type enforcement. This is a mandatory access control policy that limits what processes within a host can do. Each process is assigned a domain, and each object is assigned a type. The access control policy specifies which domains can access which types and in what ways. By carefully analyzing the needs of each process in the system, we can create firewalls within each host that ensures that compromise of one process does not lead to compromise of another. Note that this mechanism reduces the power root privilege, so that root compromises no longer provide the adversary with complete control. If an adversary does compromise some process on a host, the adversary is constrained to accessing the resources available to that process. This greatly limits the ability of the adversary to use the compromised process to achieve further compromises. Domain-type enforcement mechanisms are available on both Security Enhance Linux and FreeBSD, and similar mechanisms are available on Immunix Linux. To provide this capability on Solaris, we plan to use operating system wrapper technology.

These two mechanisms enable the system to tolerate compromises by containing the compromised processes sufficiently to prevent further damage.

Other defense mechanisms used throughout the system include (1) separation of data and management networks through use of separate LANs within enclaves and separate IPSec VPNs between enclaves; (2) network- and host-based intrusion detection, with correlation at the enclave and system level; (3) automated intrusion response for high confidence alerts; (4) privacy, integrity, authentication, and replay protection mechanisms within Survivable Spread to protect critical communications throughout the system; and (5) FormatGuard and PointGuard technology to transform component compromises into component crashes.

3.6 Survivability Grammar

Survivability Grammar ([13], [14], [15], [16], and [17]) allows one to specify the logical architecture of a system using high-level abstractions, and to automatically generate correct component configurations that implement that architecture. Thereby, large classes of configuration errors are avoided altogether. Configuration errors are a major source of security breaches and system failure. A Survivability Grammar system consists of:

- A Requirements Language for specifying the logical architecture of a system. This language is based on formalizing the notion of "correct configuration" associated with different protocols, and typically contains global constraints upon configuration parameters. The end-to-end logical system architecture can be naturally specified as conjunctions of constraints in this Language.

- A Provisioning Engine that compiles constraints in the Requirements Language into detailed component configurations. It is used recursively to compile complex requirements.

- A Diagnosis Engine that checks whether constraints in the Requirements Language are true, given definite component configurations. It is used recursively to diagnose complex requirements.

We used Survivability Grammar to generate configurations for our VPN and Spread infrastructure for the Critical Design Review (CDR) demonstration. This consisted of 42 workstations and 7 routers. See Figure 5 through Figure 7, below. To gain a sense for the usefulness of Survivability Grammar, the VPN and Spread infrastructure were specified in less than three pages. The specification was compiled into about two hundred pages of component configuration files. Changes were always made to the specification, never to the configuration files. This dramatically improved the manageability of the network. Details are available in Sections 12 and 13 in [7]. Configurations were generated for the following logical structures.

- All LANs in all enclaves. Three enclaves were set up at Telcordia and two at NAI Labs.

- A GRE tunnel linking the Cisco router backbones at Telcordia and NAI Labs through company firewalls.
- OSPF routing with MD5 over the backbone.
- Wackamole gateway routers in each management and data LAN.
- A data overlay of GRE tunnels linking data Wackamole routers.
- A management overlay of GRE tunnels linking management Wackamole routers.
- Separate RIP routing domains on the management and data overlays.
- Separate Spread networks on the management and data overlays.

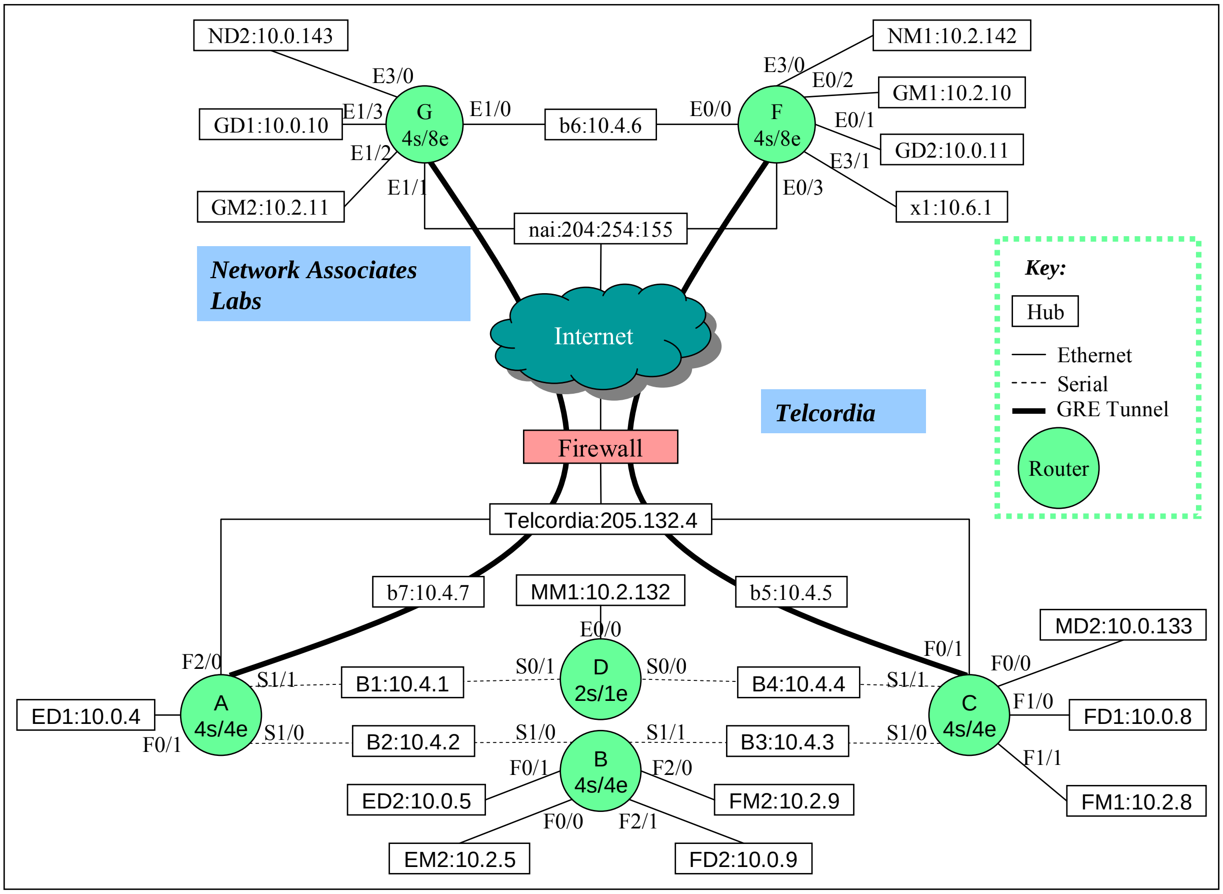

Figure 5. Joint Network Associates-Telcordia Backbone

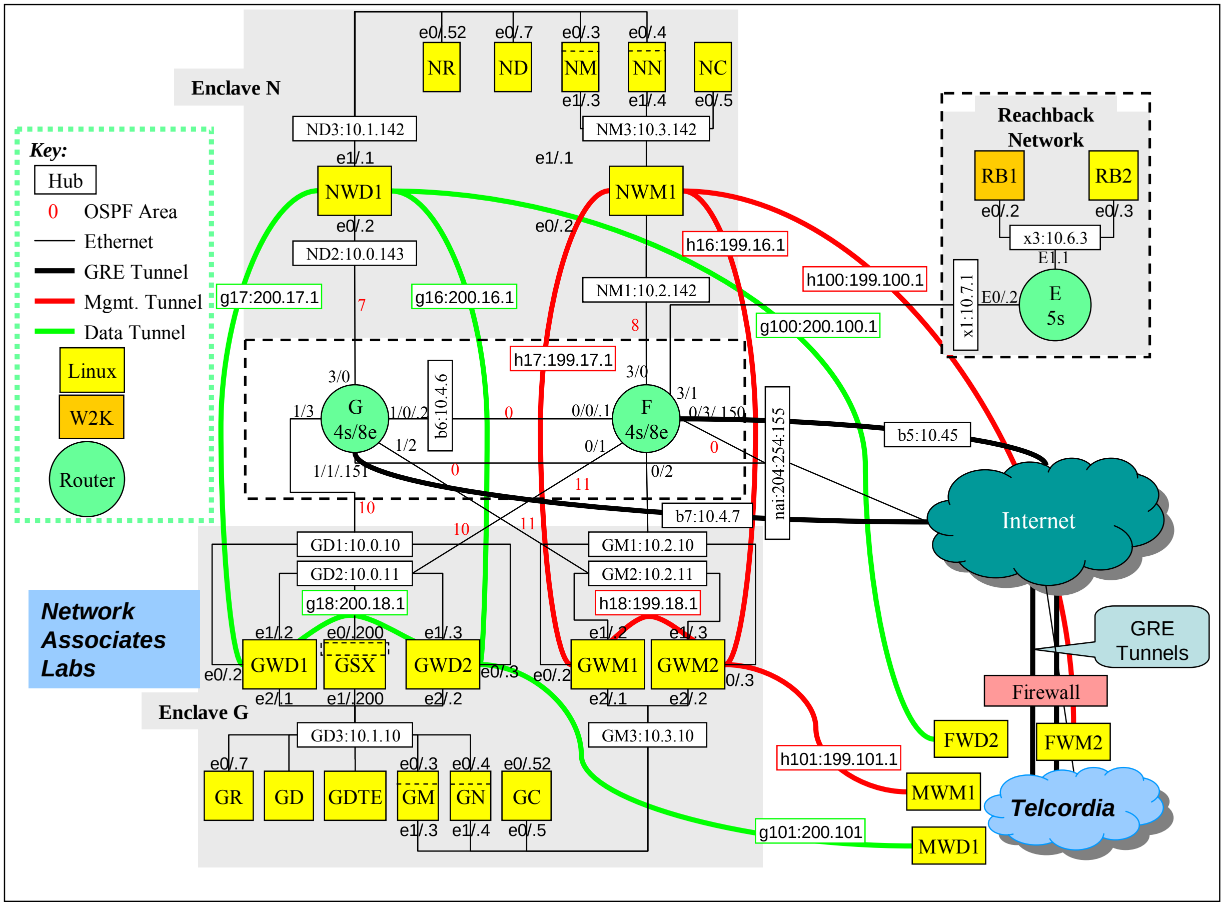

Figure 6. Network Associates Labs Configuration

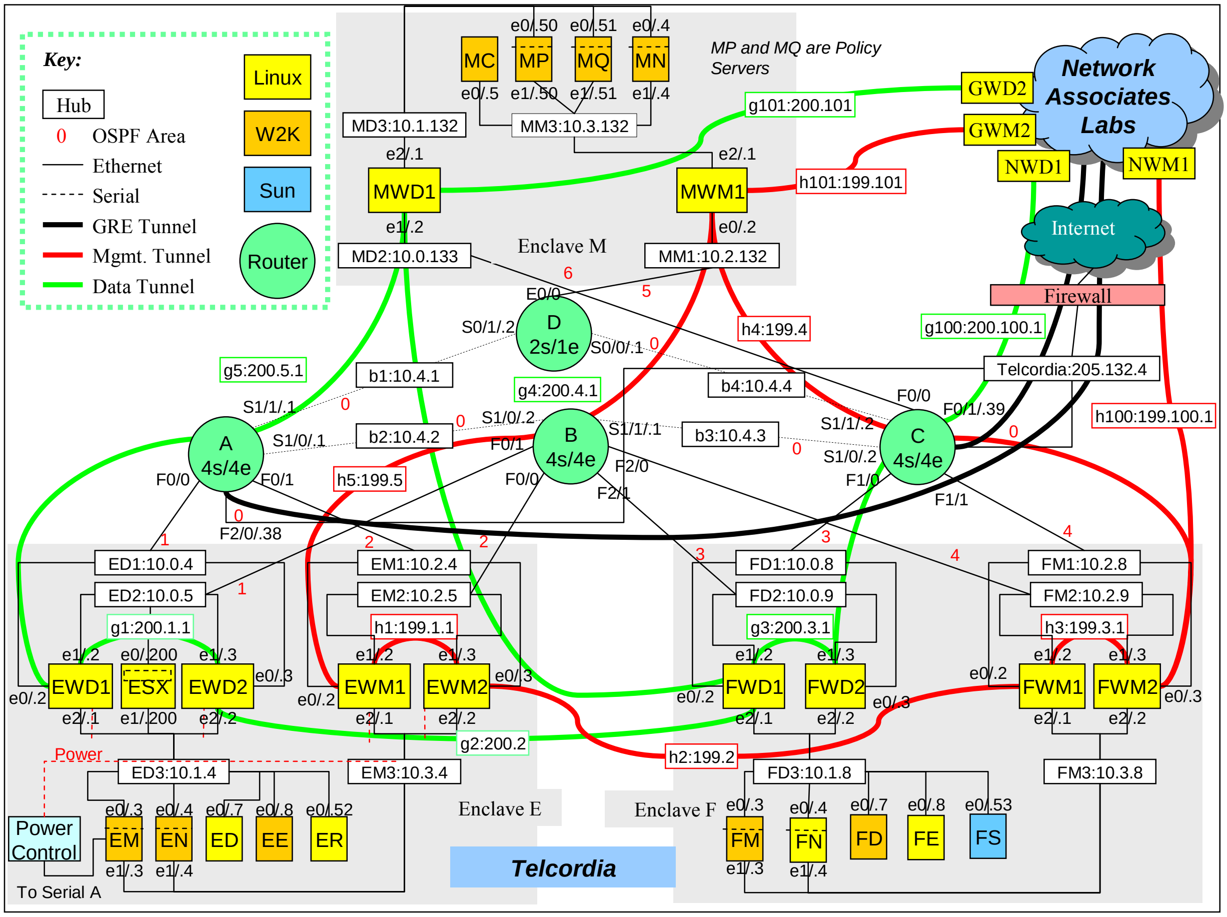

Figure 7. Telcordia Configuration

4.0 Demonstration

As part of our design effort, the Boeing Team implemented and demonstrated portions of our design. This technology demonstration was intended to reduce the development risk for a full implementation of our design. Our purpose was to provide evidence of the feasibility of constructing what we believe were initially some of the more high-risk portions of our design. Our intention was to demonstrate feasibility, not to produce a complete, full function, robust, fully intrusion tolerant implementation of the full design. The three main areas where we performed demonstrations are as follows.

- **JBI functionality supported over new infrastructure.** Our design supports the JBI over a significantly modified infrastructure. The main modifications include the use of JMS over Spread for the publish subscribe infrastructure, the use of wide area networking, the use of the SE Linux computing platform, and the use of replicated data storage (as a replacement for some JBOSS functionality). We believe that it was important to demonstrate that we could make all of these changes to the infrastructure and still support the full functionality of the JBI API, called the CAPI. For us, the critical test is the ability to run unmodified JBI clients over our implementation of the JBI and still have them function correctly. We met this test with two of our demonstrations: one a demonstration of JBI publish and subscribe functionality and the other a demonstration of JBI persistence and query functionality. In both cases, we showed the correct operation of unmodified JBI clients over our new infrastructure in a benign networking environment.
- **Recovery of selected JBI clients.** Our design does not include a general checkpoint capability because we believe that it would be extremely difficult to provide this functionality in a general way for complex, legacy clients that use the services of the JBI. Instead, we provide a customized state recovery capability that makes use of the restart semantics of many of the JBI clients. Our new recovery capability updates the state of a JBI application by replaying JBI messages that the application subscribed to before its crash. It makes use of (1) a new replicated data storage method based on Spread group communication augmented with a Replication Engine to provide data consistency in the message repository and (2) a procedure for replaying JBI messages to restarting JBI applications. Because this method of recovering JBI clients relies on the specific way in which these clients now restart, we believed that it was important to implement and demonstrate the operation of the new state recovery capability with existing JBI clients. We met this test with two of our demonstrations: one a demonstration of the robust operation of the message repository and the other a demonstration of the recovery of an existing JBI client after a crash failure.
- **Critical intrusion-tolerance and fault-tolerance technologies.** Our design integrates a number of new and existing intrusion-tolerance and fault-tolerance technologies. We have gained confidence in the efficacy of these technologies through operational experience for the existing technologies and through detailed assurance arguments for the new technologies, primarily the survivability extensions to the Spread group communication software suite. We demonstrated three of these technologies, specifically (1) the use of the existing domain type enforcement (DTE) mechanisms to provide

isolation of security domains in the face of a machine compromise in which the attacker gains root access in one domain, (2) the use of Spread and Wackamole to provide a survivable router via failover, and (3) the use of new bridging software that makes use of the Spread group communication semantics to relay authentication, authorization, and management traffic between separate Spread networks with high survivability.

5.0 Conclusion

The major contributions of our effort include:

- Demonstration of how to use an assurance argument to guide the system design for a relatively complex system.
- Development of algorithms for survivable group communication based on a hybrid approach to Byzantine fault tolerance.
- Development of algorithms for achieving reliable JBI query service where only one correct database replica is required compared to the typical ⅔ of the replicas required to be correct.
- Development of a design that leverages pervasive packet filtering and domain type enforcement to provide strong containment boundaries that limit the adversary's attack possibilities.
- Development of Survivability Grammar rules for several of the distributed algorithms used in our design to eliminate vulnerabilities caused by incorrect configuration of distributed components.

One aspect of our design is that we were able to use some common mechanisms to provide several design features, reducing the number of mechanisms that must be made to operate securely. The two primary examples of this are–

- Survivable Spread is used to provide most of the communication within the system. Only Wackamole router communication (which uses Spread rather than Survivable Spread); COTS products (which use standard protocols such as HTTPS, HTTP, DNS, SMTP, POP, NTP, and DHCP); and management of COTS components (through protocols such as SSH).
- A replication engine on the SNS is used to provide consistent database access for JBI information objects, metadata, and client checkpoint data, as well as distributing access control data.

Our design shows that intrusion tolerance can be achieved without significant performance impacts using the hybrid approach. This lesson can lead to practical application of intrusion tolerance technology. To date, Byzantine fault-tolerance algorithms have primarily been studied in academic settings and not widely used in practice because of the significant performance impact caused by the large number of additional messages required to achieve Byzantine fault tolerant agreement. The use of trusted components greatly reduces the performance impact. The protocol designed for Survivable Spread requires no additional messages beyond what is required for safe message delivery.

6.0 References

[1] The Boeing Company, *Survivable JBI Requirements Technical Report*, Boeing Document D658-11003-1, September 2002.

[2] The Boeing Company, *Survivable JBI Concept of Operations*, Boeing Document D658-11005-1, November 2002.

[3] The Boeing Company, *Assurance Argument for a Survivable JBI*, Boeing Document D658-11005-1, July 2003.

[4] The Boeing Company, *Survivable Spread: Algorithms and Assurance Argument*, Boeing Document D950-10757-1, July 2003.

[5] The Boeing Company, *Survivable JBI Preliminary Design*, Boeing Document D658-11007-1, April 2003.

[6] The Boeing Company, *Survivable JBI Risk Mitigation Plan*, Boeing Document D658-11010-1, July 2003.

[7] The Boeing Company, *Survivable JBI Detailed Design*, Boeing Document D950-10758-1, July 2003.

[8] The Boeing Company, *Survivable JBI Phase 2 Development Plan*, Boeing Document D950-10759-1, July 2003.

[9] The Boeing Company, *Survivable JBI Assessment and Validation Plan*, Boeing Document D950-10760-1, July 2003.

[10] The Boeing Company, *Analysis of Known Attacks Against the Survivable JBI*, Boeing Document D658-11011-1, July 2003.

[11] Paulo Verissimo, *Dependability, Security, two faces of a same coin?*, International Conference on Dependable Systems and Networks 2003 workshop presentation.

[12] Y. Amir and C. Tutu, *From total order to database replication*, in Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pages 494{503, Vienna, Austria, July 2002, IEEE.

[13] Qie, X., Narain, S., *Diagnosing BGP Configuration Errors with Service Grammar*, Proceedings of Usenix Systems Administrators Conference, LISA, 2003.

[14] Narain, S., Coan, B., Cheng, T., Kaul, V,, Parmeswaran, K., Stephens, W., *Building Autonomic Systems Via Configuration*, Proceedings of Autonomic Computing Workshop, Seattle, WA, 2003.

[15] Barton, M., Atkins, D., Narain, S., Ritcherson, D., Tepe, K., *Integration of IP Mobility and Security For Secure Wireless Communications*, Proceedings of IEEE International Communications Conference, New York, NY, 2002.

[16] Narain, S., Shareef, A., Rangadurai, M., *Diagnosing Configuration Errors in Virtual Private Networks*, Proceedings of IEEE International Communications Conference, Helsinki, Finland, 2001.

[17] Narain, S., Vaidyanathan, R., Moyer, S., Stephens, W., Parmeswaran, K., Shareef, A., *Middleware for Building Adaptive Systems Via Configuration*, Proceedings of ACM SIGPLAN Workshop on Optimizing Middleware, Salt Lake City, UT, June 2001.

7.0 Acronyms and Abbreviations

AFRL	Air Force Research Laboratory
ARP	Address Resolution Protocol
DARPA	Defense Advanced Research Projects Agency
DTE	Domain Type Enforcement
EFW	Embedded Firewall
GRE	Generic Routing Encapsulation
GINSU	Guaranteed Internet Stack Utilization
IDS	Intrusion Detection System
IOR	Information Object Repository
IP	Internet Protocol
IPSec	IP Security
JBI	Joint Battlespace Infosphere
LAN	Local Area Network
MDR	Meta Data Repository
NIDS	Network Intrusion Detection System
OASIS	Organically Assured and Survivable Information Systems
OSPF	Open Shortest Path First
RIP	Routing Information Protocol
SNS	Secure Network Server