CONFIDANT: Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions

Adam J. Rocke and Ronald F. DeMara
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2450

Abstract

File Integrity Analyzers serve as a component of an *Intrusion Detection* environment by performing filesystem inspections to verify the content of security-critical files in order to detect suspicious modification. Existing file integrity frameworks exhibit single point-of-failure exposures. The *Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions (CONFIDANT)* framework aims at fail-safe and trusted detection of unauthorized modifications to executable, data, and configuration files. In this paper, an IDS architecture taxonomy is proposed to classify and compare CONFIDANT with existing frameworks. The CONFIDANT file integrity verification framework is then defined and evaluated.

CONFIDANT utilizes three *echelons* of agent interaction and four *autonomous behaviors*. *Sensor* agents in the lowest echelon comprise the *sensor level* to generate an assured report to companion agents of computed MD5 file digests. At the *control level*, *beacon* agents verify file integrity based on the digests from sensor-level agents assembled over time. Upper echelon transactions occur at the *response level*. Here *watchdog behavior* agents dispatch *probe* agents to implement the alarm signaling protocol. CONFIDANT has been implemented in the Concordia agent environment to evaluate and refine its agent behaviors. Evaluation shows that CONFIDANT mitigates single point-of-failure exposures that are present in existing frameworks.

Keywords: Distributed Agent Control and Dispatch, Agent Handshaking Protocols, Network Security, Intrusion Detection System Taxonomy

† Supported in-part by National Security Agency subcontract MDA904-99-C-2642.

1 Introduction

Intrusion Detection Systems (IDSs) serve to identify security breaches of com-

puter systems and are an important component of defensive measures protecting computer systems from tampering. While intrusion detection is not a complete defense, it can play a significant role in overall network security [1]. Motivation for using intrusion detection technology varies from collecting forensic information to triggering actions in order protect computing resources. ID may also be used in order to identify and correct existing vulnerabilities [2]. *File Integrity Analyzers* serve as a component of an intrusion detection environment by performing filesystem inspections to verify the content of security-critical files in order to detect suspicious modification. When an unauthorized modification is detected, an alert is generated to notify a security administrator of potential tampering. File analyzers can also provide damage control guidance by identifying modified filesystem resources that must be restored to a known valid state.

1.1 File Integrity Operation Overview

File integrity tools use one or more cryptographically-based hash mechanisms such as SHA-1 or MD5 [3] to compute a digest checksum for monitored files. Digests are subsequently recomputed and then compared against previous values to detect if the contents of the file have been modified. Ten of the most widely installed file integrity analyzers are reviewed in [4]. One of the most popular is commercially marketed under the name *Tripwire* [5]. It utilizes a *policy file* under the control of an administrator to describe the expected behavior of system and data files. The policy file identifies files that are expected to change and the types of changes permitted to each file in order to preclude the misidentification of anticipated changes as tampering. Upon initial execution, a *baseline database* is created according to the policy file using cryptographic hash functions. Subsequent scan operation is illustrated in Figure 1. During scan operation, the policy file contents are obtained to determine the monitored system files that are to be inspected. Hash functions compute a digest checksum for the first specified file. The result is compared to the value stored in the baseline database. This process is repeated for all policy file entries. Once all specified files have been processed, a report is generated containing any discrepancies encountered during operation.

In a networked environment, *Tripwire for Servers* executes on each node [6]. The *Tripwire Manager* [7] interacts with the Tripwire servers via Secure Socket Layer (SSL). This form of centralized policy management enables an administrator to define a single policy and distribute it to many similar systems across the enterprise. Additionally, file attributes such as owner and access modes are checked for inconsistencies. Tools such as *AIDE* [8], *Veracity* [9], *integrit* [10], and others operate similarly. An overview of selected agent-based intrusion detection and file integrity frameworks is presented in [11].

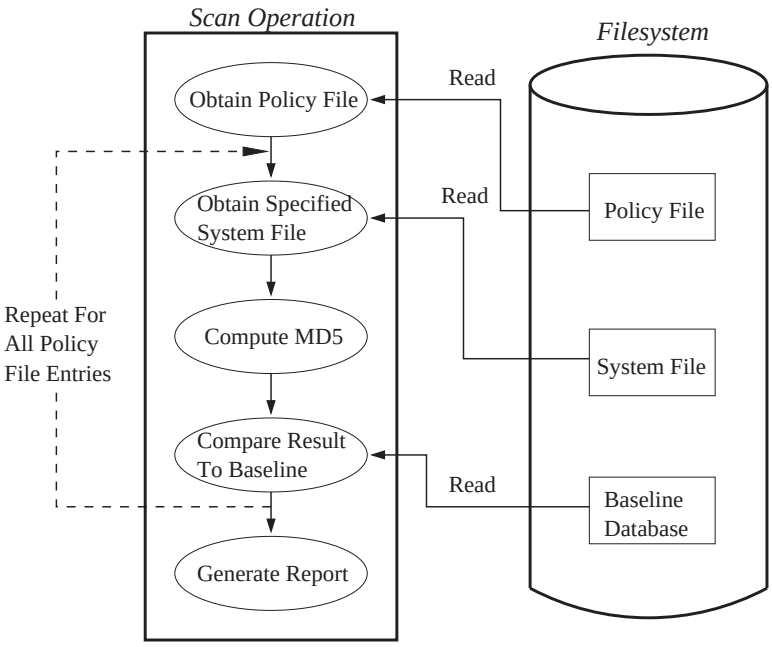

Figure 1: File Integrity Scan Operation

1.2 Using Mobile Agents for Intrusion Detection

Mobile agents can address the capacity, scalability, and efficiency limitations of existing *monolithic* IDS architectures. Whenever a new form of unforeseen intrusion is identified, a monolithic IDS has to be substantially rebuilt to handle it, which is not a straightforward task. Mobile agent systems can be better suited to achieve the following desirable characteristics for an IDS [12]:

- run continually with minimal human supervision,
- strive for incurring minimal overhead on the system where it is running,
- be able to adapt to changes in system configuration and use,
- be able to scale to a large number of hosts, and
- provide graceful degradation.

An intrusion detection system using a mobile agent framework can exhibit these characteristics. While a single agent cannot adequately perform intrusion detection since its vision is limited to a small portion of the network, multiple agents cooperating with each other can provide powerful IDS capabilities across heterogeneous resources. Agent mobility allows execution of independent instances of code in order to increase tampering complexity by requiring successful tampering to occur on distributed nodes simultaneously. Mobility also

Table 1: Advantages of Using Agents in Intrusion Detection

Configurability	Can be added or removed from a system without altering other IDS or OS components
	Compatible with heterogeneous networks
Extendability	When a new attack is identified, new agents can be developed and added
	Easier coordination between interacting components
Efficiency	Communication cost reductions
	Hide network latency
	Reduce network load
	Execute asynchronously and autonomously
Robustness	Can continue to operate in the presence of physical or logical modifications to the network environment
	Support fault-tolerant behavior
	Flexible architecture for distributed computation

provides spatial ambiguity that increases the level of difficulty by which specific routines can be altered as their physical binding is not known a-priori.

Since agents are independently running entities, they can be added and removed from a system without altering other components and consequently remove the need to restart the IDS [13]. Also, whenever any sign of a new form of attack is identified, new specialized agents can be developed, added to the system, and configured to meet a specific security policy [14]. Other advantages of using mobile agents in an intrusion detection system [15] are listed in Table 1 where Configurability and Extendability can be traded off for Efficiency and Robustness.

1.3 CONFIDANT Objectives

The *Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions (CONFIDANT)* framework aims at fail-safe and trusted detection of unauthorized modifications to executable, data, and configuration files by limiting the exposures present in existing frameworks. An exposure is defined as a weakness or vulnerability that may be exploited by malicious actions. CONFIDANT exemplifies a third-generation agent-based security framework, based upon previous experience from the TACH [16] and FICA [17] systems implemented at the University of Central Florida. In order to overcome the limitations of TACH and FICA as described in [11], CONFIDANT is designed of reduce single point-of-failure exposures in existing IDS frameworks. This goal is addressed by:

1. identifying single point-of-failure exposures in IDSs,
2. designing experiments to evaluate performance, and
3. comparing performance of the proposed and existing approaches.

In this paper, the CONFIDANT file integrity verification framework is defined and evaluated. An IDS Taxonomy is proposed in Section 2 to provide

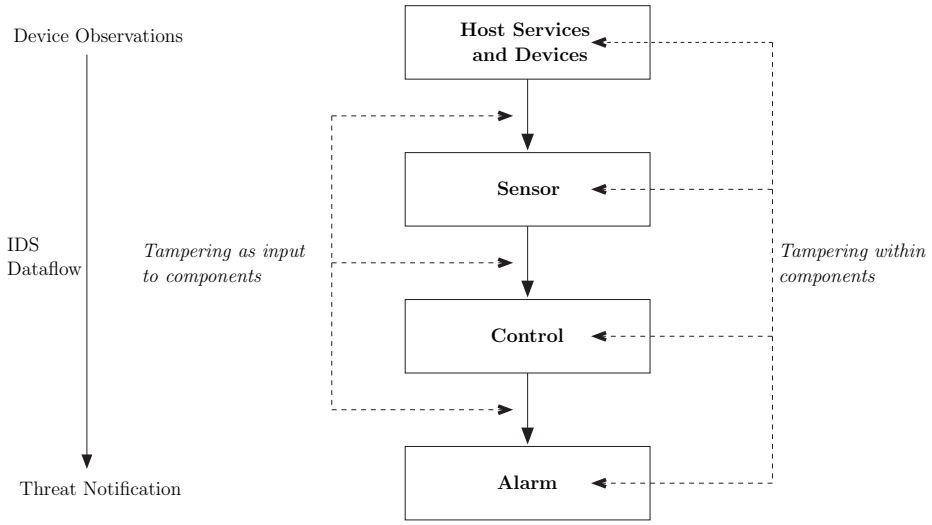

Figure 2: IDS Conceptual Architecture

architecture comparison of CONFIDANT to other frameworks. Section 3 defines the CONFIDANT framework and design methodology including behaviors and interaction of agents. Example dispatch and communication scenarios are provided. CONFIDANT evaluation methodology and results are provided in Section 4.

2 IDS Vulnerabilities and Taxonomy

An IDS can be described in terms of *sensor*, *control*, and *alarm* logical components, as illustrated in Figure 2. Sensor components gather raw data from the system environment such as the contents of files being analyzed. Control mechanisms provide logic and decision-making routines to interpret the sensor outputs. Alarm subsystems respond to violations by implementing alert conditions. Tampering can occur at the host services and devices level, within each logical IDS component, and as input to each component. A review of single point-of-failure exposures in existing agent-based intrusion detection and file integrity frameworks is presented in [11]. Attack pathways capable of corrupting an ID framework, or *tampering modes*, are also defined therein.

Based on the previously described exposures, certain IDS design characteristics are utilized by CONFIDANT to meet the design goals. For instance, tampering complexity increases if functionality is not bound to a physical location. One technique to assist in avoiding a single point-of-failure exposures is to distribute IDS control across network nodes so that compromising a single node will not fatally disrupt execution on other nodes. Agent mobility is employed to serve this purpose. Based on these characteristics, the following taxonomy

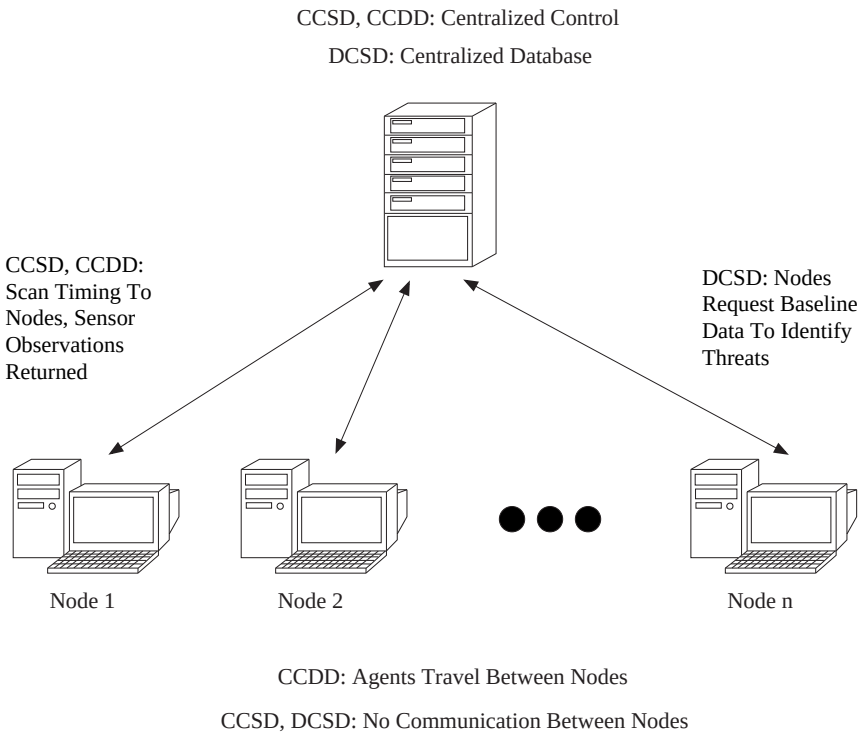

Figure 3: Control and Dispatch Strategies of IDS Architectures

of IDS architectures is proposed.

2.1 Centralized Control, Static Dispatch Architectures

Centralized Control with Static Dispatch (CCSD) architectures are the most fundamental configuration of IDSs. Architectures of this type restrict data and control functionality to a single machine and have a static network topology as illustrated in Figure 3. An example of this category is any single-host IDS, such as Tripwire or AIDE execution on a single machine with a local baseline database. Here the single machine controls the IDS and all data remains local to each machine. If an intruder is able to compromise the single machine, the IDS can be compromised. An example of a multi-host CCSD IDS is the Tripwire Manager controlling multiple instances of Tripwire for Servers.

2.2 Distributed Control, Static Dispatch Architectures

In order to overcome limitations in CCSD architectures and provide functionality for multiple network nodes, control functionality and data can be distributed across a network domain. *Distributed Control with Static Dispatch (DCSD)* ar-

chitectures are more sophisticated than CCSD due to the inclusion of multiple machines with control responsibilities in a distributed network, as shown in Figure 3. IDS control is distributed as individual machines are responsible for local computations. The dispatch of information is static as a central management database node records data served by individual nodes. Data dispatch is static due to a well-defined network topology and communication between client nodes and the central database. If an attacker is able to control the centralized management database, the entire IDS can be compromised. It is possible that the distributed management consoles are unable to receive accurate signature data and alerts generated by the client nodes, so DCSD architectures of this type exhibit a single point-of-failure. Hummingbird [18] is an example of a DCSD architecture as it maintains a database for storing misuse data. *Cooperating Security Managers (CSMs)* [19] are another example of a DCSD architecture. While CSM does not maintain a misuse database, interaction between CSMs on each host is static.

2.3 Centralized Control, Dynamic Dispatch Architectures

Centralized Control with Dynamic Dispatch (CCDD) architectures have a single control module coupled with dynamic data movement. An example of this type of architecture is FICA [17]. In the FICA framework, agents are dispatched across the network to collect data and return only the filtered results to a single host for processing. This single host serves as the centralized control node as illustrated in Figure 3. Since the agents can be sent to any node in the network at times specified by the server during runtime, there is a dynamic dispatch of agents. There is also a dynamic dispatch of data as data travels with the agents throughout the network domain. As with the previous two types of architectures, a single point-of-failure vulnerability exists. When client nodes rely on a single control server, modifications to control instructions can alter scan timing and produce erroneous scan results. In the case of mobile agents, whenever agents report to a single physical address that becomes compromised, resources are no longer protected. The hierarchical structure of distributed IDS elements such as agents, transceivers, and monitors is subject to compromise.

2.4 Distributed Control, Dynamic Dispatch Architectures

The final architecture defined is *Distributed Control with Dynamic Dispatch (DCDD)*. CONFIDANT exhibits an architecture of this type. This novel architecture class diffuses single point-of-failure exposures common to the other architectures by distributing control of the IDS, as well as providing a dynamic dispatch of data contained in agents. Control and data are distributed across a completely connected network domain. By distributing both data and control functionality, the single point-of-failure in other architectures is mitigated. Mitigation is defined as a reduction in severity or avoidance of otherwise negative consequences.

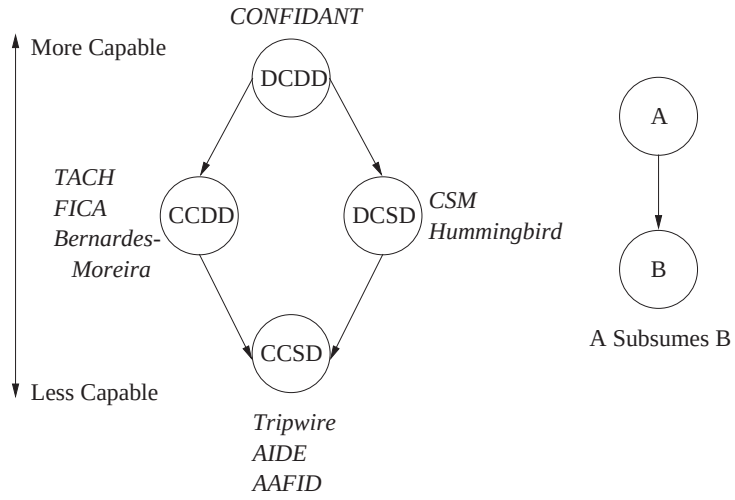

Figure 4: Hierarchy of Architecture Classes

A hierarchy of the architecture classes defined in this section is given in Figure 4. As shown in the figure, class A is said to subsume class B if the mechanisms in class A can perform the operations of those in class B. For instance, when FICA (CCDD) either doesn't dispatch agents or dispatches agents to the local machine, it operates as a CCSD architecture. A DCDD architecture such as CONFIDANT can function as a CCDD architecture if agents are required to report to a single centralized host. If a DCDD architecture can be realized, then it will provide a general technique for intrusion detection capable of mitigating additional vulnerabilities. Section 3 will describe how a DCDD architecture is realized in CONFIDANT. To assess the performance of CONFIDANT's DCDD realization, a number of comparative metrics are developed in the following section.

3 CONFIDANT Agent Framework

CONFIDANT extends the TACH and FICA frameworks described in [11] based upon the requirements listed in Table 2. Requirements are listed in order of the progression of IDS processing. Requirements R1, R2, and R3 define the required properties of the environment in which the agents operate. Requirement R4 dictates the security of the agent transactions themselves. These critical requirements involve the integrity of the agent execution, the inter-agent communication, and the processing results maintained during execution. The corresponding operating assumptions OA1 – OA4 ensure that CONFIDANT agents are able to verify the correctness of the individual hosts on the network as well as other agents. Also, file data obtained by agents must be accurate. The stated assumptions coupled with use of mobility significantly diminish the ability of

Table 2: CONFIDANT Operating Assumptions

Requirement		Operating Assumption	
R1:	File information obtained by agents is legitimate	OA1:	Agents have direct disk access to ensure accurate file validation
R2:	Scan timing is not predictable	OA2:	Agents perform filesystem locking upon arrival
R3:	Agents execute in a protected environment	OA3:	Initial configuration is well-formed and completely installed
R4:	Agent interactions are robust	OA4:	Agent transport and communication occur via SSL to preclude spoofing

any attacker to *simultaneously* modify every agent in order to compromise file integrity capabilities before an alarm is sounded elsewhere in the network.

3.1 Agent Structure

The general structure of a CONFIDANT agent is shown in Figure 5. CONFIDANT agents contain executable *behavior code* and a *persistent data repository* to store file integrity data collected while traversing the network. The agent's *itinerary* can be updated dynamically to determine the agent's route in real-time to help preclude tampering. An *agent gateway* resides on each physical processor and provides services required by the agents including *ID*, *mobility*, and *communication management* to mitigate IDS tampering modes as described below.

3.2 Agent Gateway

The CONFIDANT *agent gateway* provides the interface between the host services and the agents. It also maintains network communication channels for agent dispatch to remote nodes and agent interaction. In order to allow agents to travel to remote nodes, a gateway must be executing on every monitored host in the network. Each gateway, $G = \{g_1, \ldots, g_n\}$, is a monitored network node.

Each mobile agent gateway:

- creates and disposes of application agents,
- monitors local agents executing on a host,
- receives reports generated by remote agents, and
- forwards reports generated by local agents to remote gateways.

Tampering at the gateway level remains a potential vulnerability. To maintain IDS capability, the gateway needs to be protected from tampering and disabling. This is addressed by the assumptions listed in Table 2, and is a topic of current research [20] [21] [22]. Disabling a CONFIDANT gateway results in alarm notification when either an agent attempts to travel to the gateway or an attempt is made to communicate with an agent that was residing on the gateway when it was disabled.

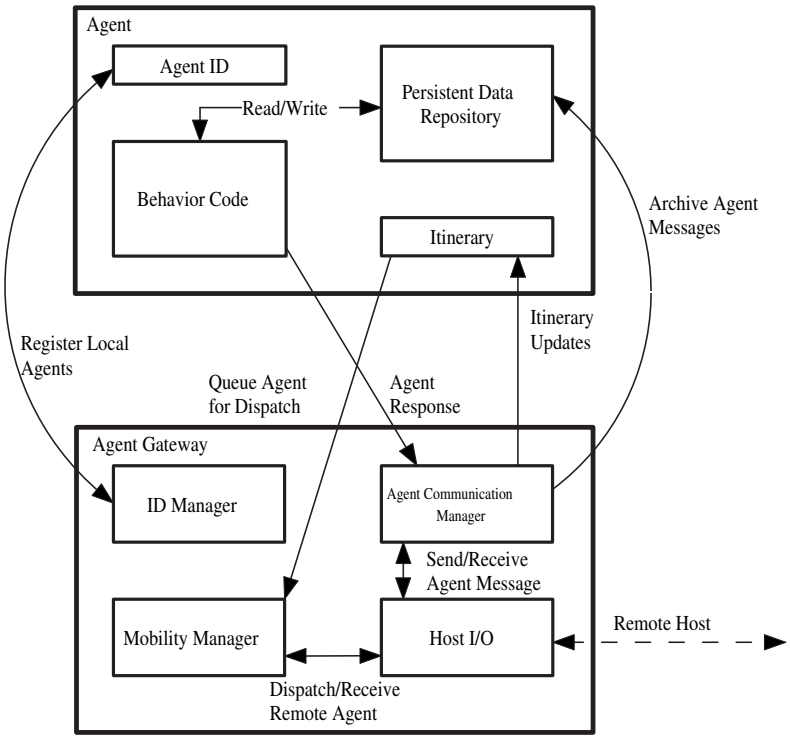

Figure 5: CONFIDANT Agent Structure [11]

Table 3: CONFIDANT Agent Behaviors

Behavior	Life span	Execution Cycles Per Gateway Visit
Probe	One Execution Cycle	Exactly One
Sensor	Infinite	Exactly One
Beacon	Infinite	Multiple Based On Time Intervals
Watchdog	Infinite	Multiple Based On Target Criteria

3.3 Autonomous Behaviors

The mobile agent framework makes use of four distinct types of agents, each defined for a specific *life span* and *execution cycle*. An agent life span is the duration from initial dispatch until execution is complete according to its *itinerary*. An agent execution cycle includes agent-specific processing and related messaging. For instance, the execution cycle for an agent that computes file MD5 values is simply reading a file and calculating the MD5 which occurs once per gateway visit. The execution cycle for an agent that correlates MD5 values is to receive and process a message from a MD5 computation agent. Based on messages received, multiple event cycles can occur on a single gateway visit.

Every CONFIDANT agent behavior extends the general agent structure illustrated in Figure 5. Individual behaviors are defined in the following subsections and summarized in Table 3. Each agent has a specific behavior based on lifespan and execution cycles per gateway visit. All agents regardless of behavior, perform the same dispatch and communication functions described below.

Probe

The *Probe* agent behavior is the most basic type of CONFIDANT agent. Its life span is only a single execution cycle. A Probe agent is dispatched to the destination gateway, an arrival confirmation message is sent, and the agent is terminated once execution is complete. Probe agents are primarily used for sounding alarms and sending messages to other remote gateways.

Sensor

Sensor agents, like Probe agents, have a single execution cycle per gateway visit, but differ in that their life span is perpetual. One role of Sensor agents is to inspect for changes to the network topology, such as hosts added or removed from the network.

Beacon

Beacon agents have an infinite lifespan and can have multiple execution cycles per gateway visit. As the name implies, beacon execution cycles occur at regular time intervals. Since Beacon agents report on regular intervals, several cycles may occur while the agent resides on a single gateway.

Watchdog

As is the case with Beacon agents, *Watchdog* agents have an infinite life span and multiple execution cycles per gateway visit. Unlike Beacon agents, however, Watchdog agents perform execution cycles based on a certain target criteria. For instance, a Watchdog agent is used to monitor agent filesystem scan results. When a discrepancy is encountered a Watchdog agent dispatches Probe agents to specific nodes to sound the alarm.

3.4 Agent Echelons

CONFIDANT agents operate in distinct echelons as shown in Figure 6. In simple domains with limited overhead, functionality can be encompassed by Watchdog agents to perform filesystem scans and Probe agents to send alarm notification to remote gateways. As network complexity and overhead increases, functionality can be distributed into distinct functional components as described below.

The *Sensor level* shown in Figure 6, is responsible for *surveillance*. In this level, *Sensor* agents are dispatched in order to obtaining file signatures and information regarding network topology. State detection sensor agents monitor the network for changes in gateway status. For instance, if the power is removed from a particular host and the gateway becomes unavailable, state detection agents will signal the network topology agents to make appropriate itinerary updates with other agents. File inspection sensor agents are responsible for visiting agent gateways and obtaining file signatures to transmit to the data correlation module.

The middle echelon is the *Control level* which provides updates for agent itineraries as well as performs result collection and correlation. The network topology module is responsible for obtaining network status information from the sensor echelon and relaying appropriate updates to the file inspection, data correlation, and alarm dispatch modules. The data correlation module receives MD5 signatures from the file inspection module at the sensor echelon. Here data is collected and analyzed to provide error information to the alarm dispatch module. Both the network topology and data correlation modules utilize the Beacon agent behavior.

The final echelon is the *Response level* and consists of the alarm dispatch module. Watchdog agents are used to collect and distribute alert notification messages from the control level to remote gateways. While alarm messages can be sent to remote agents, gateways that are not currently hosting agents are unable to receive messages directly. In order to overcome this limitation, Probe agents are dispatched by the alarm dispatch module to transport appropriate alarms. Probe agents travel to the remote gateway, reply with an acknowledgment to confirm arrival, and perform alarm notification routines. Encapsulation of alarm messages within an agent enables alarms to be signaled on any gateway on the monitored network.

Reducing the number of false alarms generated by an IDS is of particular importance. An increasing number of alarm notifications can overwhelm a secu-

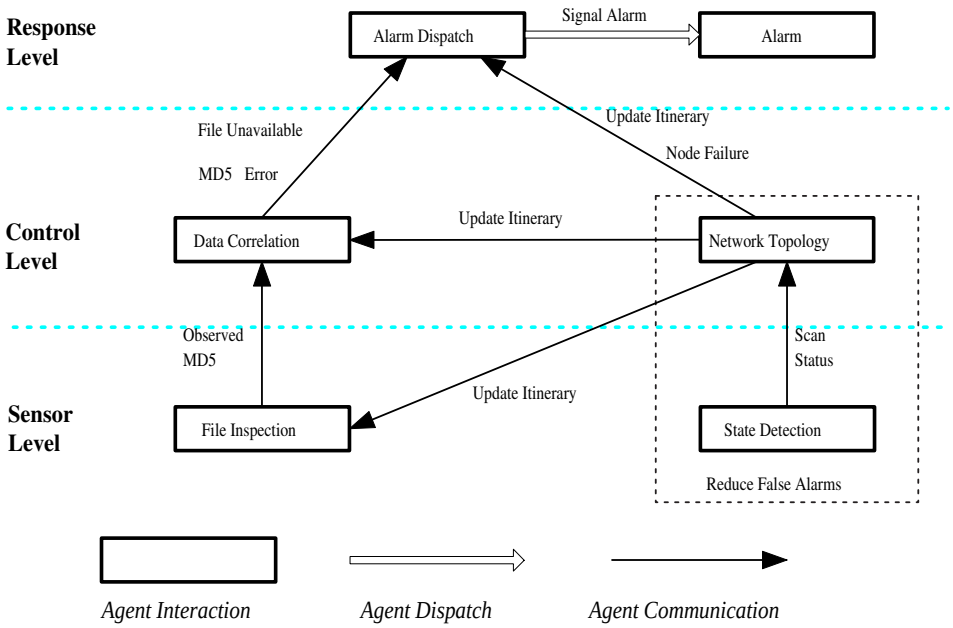

Figure 6: CONFIDANT Echelons

rity administrator and possibly reduce the perceived severity of future alarms. The Network Topology and State Detection modules in Figure 6 are designed specifically to reduce redundant alarms. Consider a subnet of gateways that are subject to frequent resetting, as in a student computer lab. Once gateway termination is detected, agents signal alarms to warn of tampering. State Detection agents relay gateway status to Network Topology agents. Messages are subsequently transmitted to other agents in order to provide itinerary update data. If potential tampering is detected and appropriate alarms triggered, agents can omit future scans and reduce the number of alarms.

3.5 Agent Interaction

CONFIDANT agents operating across the defined echelons are in frequent communication with each other. Groups of similar agents are divided into *committees* in order to provide coverage of a network domain. A single committee C^i is defined as $C^i = \{a_1^i, \ldots, a_n^i\}$ where i is the committee index and n is the number of agents in committee C^i. The set of all committees within a monitored network is defined as $C = \{C^1, \ldots, C^m\} = \bigcup_{i \in m} C^i$ where m is the total number of committees. Adjacent committees share common agents in order to enhance scalability and adapt to physical network layout. Due to the overlapping nature of adjacent committees, $\bigcap_{i \in m} C^i \neq \emptyset$.

Agents Independent of Committee Interaction

Some agents, specifically Probe agents, are not members of a committee and are considered independent. Handshaking interlocks provide assurance that messages sent to remote agents are received. Independent agents provide robust communication, particularly alarm messages, to remote gateways that are not hosting agents at the time the message is generated.

Independent agents are used to provide robust communication to monitored network nodes that are not currently hosting agents. The set of independent agents, $I = \{a_1^I, \ldots, a_n^I\}$, includes all CONFIDANT agents that are not members of any committee. An agent j that is a member of committees i and k is denoted $a_j^{i,k}$, and has the following properties

- $a_j^{i,k} \in C^i$,
- $a_j^{i,k} \in C^k$, and
- $a_j^{i,k} \notin I$.

Thus, the set of all CONFIDANT agents, A, is defined as $A = I + \bigcup_{i \in m} C^i$.

Interaction Within Committees

An agent *committee* is a group of related agents that operate across a division of the network domain and communicate via broadcast messages to other members. Members within a committee are bound to a physical subset, $g \subset G$, of the monitored network. Table 4 lists eight agent event types. These events are used to provide interlocked transactions so that agents are aware of both status and location of other committee members. Interlocking provides robust agent communication in order to ensure messages are received by remote committee members across the network domain. The ability to monitor remote agents is a requirement of the CONFIDANT design.

Figure 7 illustrates communication between agents within a committee. Three agents are shown. Two agents a_1^1 and a_2^1 from committee C^1 and a_1^2 from C^2. Agents can receive messages only from other committee members. Consequently, agent a_2^1 can receive messages from agent a_1^1, but not from agent a_1^2.

Interaction Between Committees

Overlapping committees are used in order to take advantage of physical network domain configuration and enhance scalability. Agents within a committee maintain peer-to-peer connectivity within their group as illustrated previously. The common nodes between adjacent committees enable monitoring of different portions of the network.

Figure 8 illustrates communication between agents where one is a member of two overlapping committees. In this example, agent a_1^1 is a member of committee C^1 and agent a_1^2 is a member of committee C^2. Agent $a_1^{1,2}$ is a member of both C^1 and C^2 and consequently enables communication between both committees.

Table 4: CONFIDANT Agent Event Types

Event Name	Description
`AgentArrived`	Sent when a dispatched agent arrives at a remote gateway
`AgentArrivedACK`	Sent to a recently dispatched agent upon receiving arrival notification
`AgentTravelRequest`	Sent by an agent to Committee members when prepared for dispatch
`AgentTravelProceed`	Response from Committee members when dispatch is acknowledged
`MD5OK`	Sent upon valid MD5 computation
`MD5Error`	Sent upon modified MD5 computation
`MessageACK`	Sent to confirm message receipt
`AgentUnavailable`	Sent to Committee members when a member does not respond to communication
`HostUnavailable`	Sent to Committee members when travel to a gateway is prohibited

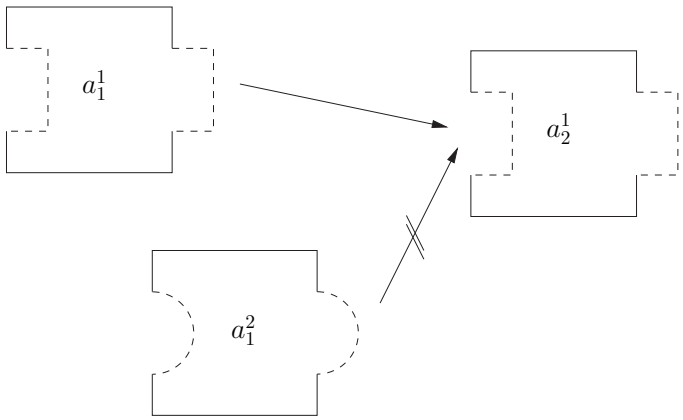

Figure 7: Agent Communication Within A Committee

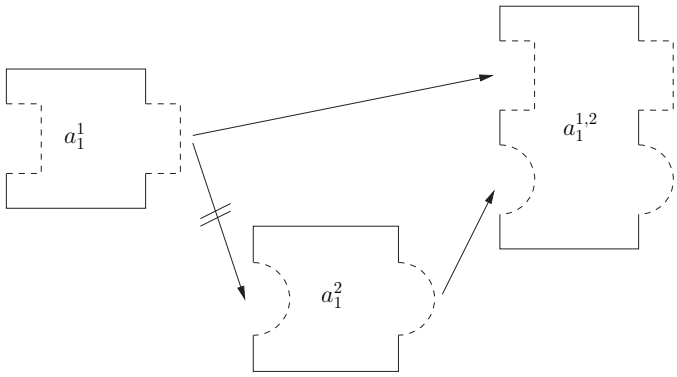

Figure 8: Agent Communication Between Overlapping Committees

Here agent a_1^1 is unable to communicate directly to agent a_1^2, but both are able to communicate with members of the other committee due to the overlapping nature of agent $a_1^{1,2}$.

3.6 Agent Dispatch and Communication

In order to describe dispatch and communication between agents, dispatch and communication terms are defined in Table 5 and functions are defined in Table 6. For example, a dispatch operation is denoted by:

$$a_j^i.dispatch(g_k) < delay, \Delta t_d > .$$

Consider a Watchdog agent a_4^6 in the Alarm level creating a Probe agent immediately to dispatch to gateway g_2 in order to warn of an file modification. This operation is denoted:

$$a_4^6.spawn(probe,\ MD5Error,\ g_2) < 0, \Delta t_d > .$$

The *operation* delay of 0 signifies that the operation is to take place immediately. The maximum delay allowed for agent dispatch is defined as Δt_d. All Δt terms in Table 6 define a waiting period for operation confirmation. In the previous example, if an arrival acknowledgment message is not received from the Probe agent created by the spawn operation within Δt_d, an error has occurred and is handled accordingly.

Next, consider an agent a_2^2 preparing to travel to a remote gateway by sending a travel request message to other members of committee C^2 after a 4 second delay. The corresponding operation is:

$$a_2^2.sendmsg(C^2,\ AgentTravelRequest) < 4,\ \Delta t_s > .$$

An example of agent dispatch and communication illustrating use of these definitions and functions is provided in below.

Table 5: CONFIDANT Agent Dispatch and Communication Terms

Term	Definition
`listener`	The listening agent or committee.
`msg`	The message transmitted between agents.
`delay`	The time between when a particular operation is specified and when it is performed. Delay=0 means to perform the operation immediately.
Δt_s	The waiting period from when a message is sent until a confirmation is received.
Δt_r	The time allowed to wait for an expected message to be received.
Δt_d	The time allowed between agent dispatch and the related arrival message.

Table 6: CONFIDANT Agent Dispatch and Communication Functions

Description	Definition
Creation of a new agent	spawn(b_i, msg, g_i)<delay, Δt_d >
Agent travel to remote gateway	dispatch(g_i)<delay, Δt_d >
Transmitting a message with no expected response	sendmsg(listener, msg)<delay>
Transmitting a message and waiting for a response	sendmsg(listener, msg)<delay, Δt_s >
Listening for a message	receivemsg(sending agent, msg)< Δt_r >

3.7 CONFIDANT Handshaking Scenarios

At time $t < 0$, the monitored network is not connected to the outside network. All gateways, $g_i \in G$, are operating and all agents in committees, $C^j \in C$, are executing. The network is in a safe state and CONFIDANT is operating on all monitored nodes. Sensor agents are dispatched in the sensor/inspection layer to perform file inspection and state detection operations. Beacon agents are deployed in the control layer to correlate responses from the sensor level and provide alert data to the response echelon. Watchdog agents in the response echelon await alert notification from the control level agents.

At time $t = 0$, network monitoring commences. File Sensor agents continue to traverse the network updating MD5 values for correlation by beacon agents in the control layer. While message digests are being collected and correlated, State Detection agents traverse the network scanning for any changes in resource availability. If a particular gateway is subject to frequent downtime, the itinerary of other agents are updated in order to prevent frequent attempts to visit an unavailable host. State Detection agents investigate previously unavailable nodes in order to update the Network Topology module in the control layer that resources are again available and to update agent itineraries accordingly.

Agent dispatch and communication is illustrated in Figure 9. In this example agent a_1^1 is dispatched to host g_2. Committee C^1 agents are notified and respond during a_1^1 dispatch. Dispatch and communication use handshaking with acknowledgment to ensure agents are aware of other members of their committee. Each agent a_i^1 first sends an intent to travel message and waits for

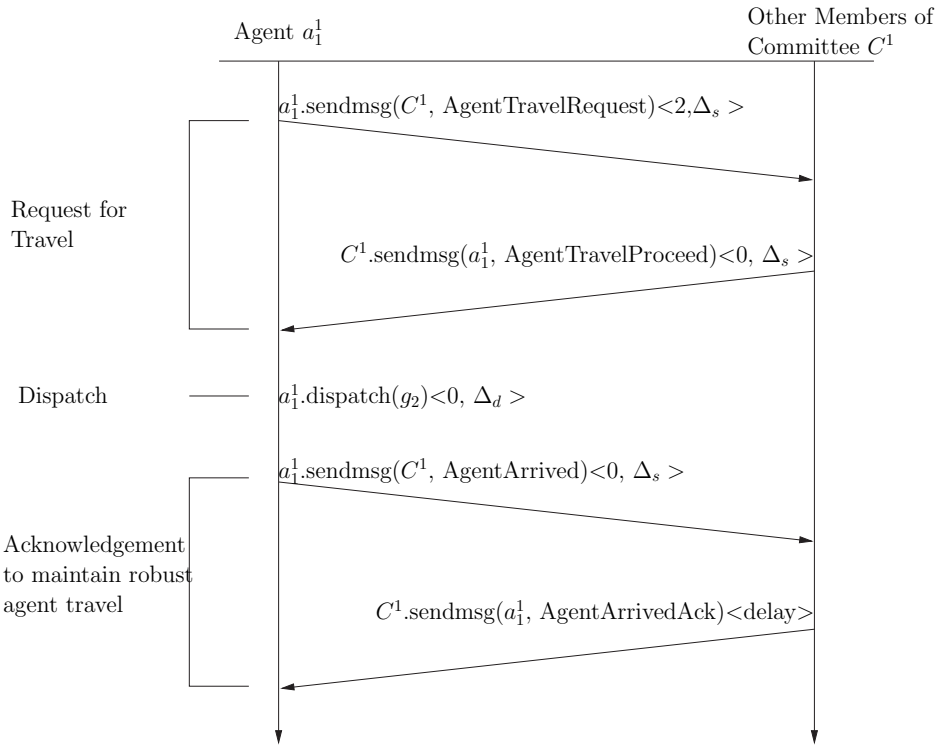

Figure 9: Dispatch and Communication Example

confirmation prior to dispatch. Upon arrival on the remote gateway, an arrival message is sent to all committee members $a^1_{j\neq i}$. The committee members then respond with an arrival acknowledgment message. Dispatch is then complete.

Handshaking is essential to guarantee that CONFIDANT agents can communicate after being dispatched to remote nodes. Consider agents not maintaining communication with other members of the group. If the agents are allowed to occupy the same host at the same time, failure of that gateway would be a single point-of-failure. Distributing redundant mobile agents in multiple interactions and levels ensures that no single agent, behavior, interaction, or level is completely responsible for IDS operation. Consider a single committee of n agents. Disabling any $x < n$ agents results in $n - x$ agents creating alarm reports stating that communication has been disrupted. Consequently, a successful CONFIDANT adversary would need to tamper with every node in the monitored network simultaneously. Additional evaluation details are provided below. In order to provide redundancy, at least $n = 2$ agents of a specified behavior must be dispatched at each level and interaction. The optimal number of agents dispatched at any level or interaction is currently being assessed as it depends on network characteristics such as topology, latency, and size. This redundancy helps ensure that agent termination or node failure does not

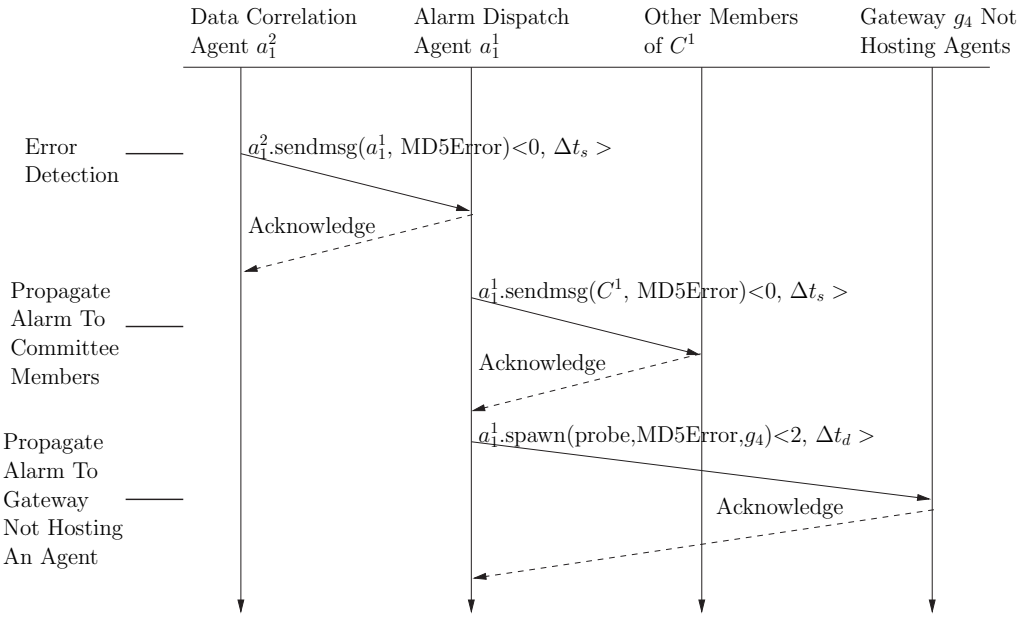

Figure 10: Propagation of Alarm Notification

compromise CONFIDANT operation.

If at any point a resource is unavailable or an observed file MD5 value differs from the baseline, Beacon agents in the control layer send alert notification to listening Watchdog agents in order to notify administrators of the modification. Figure 10 illustrates detection of a file modification. A Data Correlation agent, a_1^2 determines that the obtained MD5 value does not match the expected result. An MD5Error message is sent to an Alarm Dispatch agent to propagate the error to other nodes. Each message has a subsequent acknowledgment to provide interlocking. The error message is relayed directly to other members of committee C^1. In this example, gateway g_4 is not hosting agents and therefore can't receive messages directly. Agent a_1^1 spawns a probe agent after a two-second delay in order to transport alarm notification to the remote gateway. These interactions have been implemented on the Concordia mobile agent framework.

4 Evaluation Methodology and Results

The CONFIDANT evaluation network is composed of four physical hosts running a combination of Linux and Windows connected to a single switch. Quantitative tests involve a single committee C^1 of Watchdog agents a_i^1 traversing the network domain. Functionality of all echelons defined previously is encompassed by up to two committees.

CONFIDANT is designed to mitigate any single points of failure by em-

ploying mobile agents to realize a DCDD architecture. Currently existing IDS frameworks described in [11] exhibit a single point-of-failure by which IDS functionality is defeated if, in the worst case, a single network node is tampered with. The AAFID architecture documentation even states that a disadvantage of the framework is the existence of a single point-of-failure [23].

The agent interaction serves in part to ensure that all agents within a committee C^i do not reside on the same host. If at any time committee agents were to reside on the same physical host and that particular gateway or associated network transport mechanisms were to fail, then CONFIDANT would generate a false negative intrusion response. For the set A of agents and the set G of gateways, agent interlocking ensures that multiple agents do not occupy the same gateway when $|A| < |G|$. When $|A| \geq |G|$, agents are allowed to travel to a gateway currently hosting agents if all permissible gateways are occupied.

Termination of a gateway is detected by remote agents. If a gateway is hosting an agent when terminated, the agent is also terminated thereby disabling some IDS capability. Disabling of all agents causes a false negative response as CONFIDANT is no longer executing. The following discussion assumes that only gateways hosting agents are terminated as this is the worst case scenario. Termination of $n = |A|$ gateways results in termination of all agents and disabling of CONFIDANT. For the case in which each gateway hosts a single agent, consider a committee C^1 of agents operating in a network of gateways where $|A| = |G|$. Upon agent dispatch, one agent will travel to a node currently hosting an agent, thus resulting in one gateway hosting 2 agents, another hosting 0 agents, and the remaining $|G| - 2$ nodes each hosting a single agent. With communication interlocking ensured such that agents do not congregate to a limited subset of network nodes, $n = |A| - 1$ nodes must be tampered with within time Δt_r in order to terminate CONFIDANT monitoring capabilities. In order for an attacker to force a false negative result from CONFIDANT operation, $n \geq |A| - 1$ gateways must be terminated within time Δt_r.

Assume by contradiction that disabling $n < |A| - 1$ gateways will disable CONFIDANT operation. In this case, a minimum of $(|A| - 1) - n$ agents remain as some gateways may not have been hosting agents at the time of termination. The remaining executing agents will continue IDS operation. In the worst case, no remote agents or gateways are able to be contacted; however, local agents still continue to present alarm notification on the local gateways.

A series of tests is performed with agents executing on $|G| = 12$ logical nodes. Operation is verified by monitoring agent messages displayed on the console. Node failure is simulated by terminating the agent gateway process. Termination of a gateway where agents reside also terminates those agents. The minimum number of nodes required for CONFIDANT failure is illustrated in Figure 11.

At time $t < 0$ all agents reside on unique gateways. Agents are dispatched at time $t = 0$. Here a single agent a_1^1 travels to a gateway hosting another agent a_2^1 as described previously in order to prevent deadlock. The first gateway terminated is the one hosting a_1^1 and a_2^1. At that point, the remaining $|G| - 1 = 11$ operational gateways are hosting $|A| = |G| - 2 = 10$ agents. As subsequent

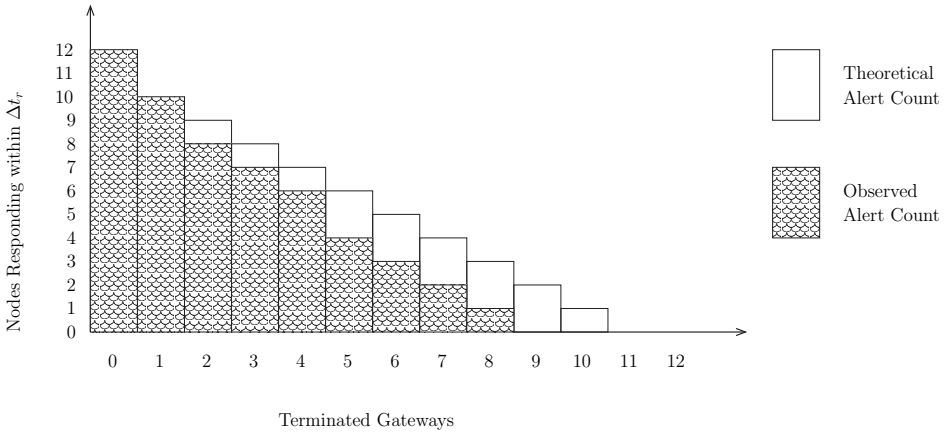

Figure 11: Agent Response in the Presence of Gateway Failure

gateways fail, agents $a^1_{j\neq 1,2}$ are unable to access terminated members of committee C^1 and are unable to travel to a disabled gateway. Each case results in alarm notification from the remaining agents. During testing, the observed alert count decreases more rapidly than the theoretical alarm count. This is due to approximately one in three agents attempting to travel to a terminated gateway not accurately reporting an alarm upon the expiration of Δt_d. Evaluation shows that CONFIDANT agents continue to traverse the monitored network domain, thus CONFIDANT does not exhibit the single point-of-failure present in existing frameworks.

5 Conclusion

Intrusion detection systems serve to identify security breaches capable of compromising computer system resource or service integrity. File integrity analyzers are a subset of host-based intrusion detection systems that verify computer filesystem data. Table 7 lists the contributions made to fields of IDS design resulting from this paper. Existing frameworks inspected in [11] were evaluated to identify IDS vulnerabilities resulting in an architectural taxonomy of IDSs. In response to the identified vulnerabilities, the CONFIDANT framework was designed specifically to mitigate the single point-of-failure in existing tools by employing four agent behaviors interacting across multiple echelons.

CONFIDANT design and testing assumptions are based on the security of mobile agents. Mobility is required to significantly diminish the ability of an attacker to *simultaneously* modify every agent or gateway in order to compromise file integrity capabilities before an alarm is sounded elsewhere in the network. The Δt_s, Δt_r, and Δt_d configuration parameters are selected in conjunction with spatial and temporal agent ambiguity such that simultaneous modification of distributed physical recourses is difficult or impossible. Agent mobility may

Table 7: Contributions of Research

Application Area	Technical Challenge	Approach Taken	Results
IDS Design	Understanding IDS relative capabilities	Defined architectural taxonomy of IDSs	CCSD, DCSD, CCDD, and DCDD categories
Mobile Agent Approach to ID	Existing frameworks exhibit a single point-of-failure	Defined a framework of behaviors and agent interaction	Sensor, control, and response echelons were developed and evaluated

introduce new vulnerabilities, however, security of mobile agents is a topic of current research [20] [21] [22]. It is critical that:

- deciphering of encrypted communication remains secure at channel endpoints,
- an agent is able to verify that the gateway being visited has not been tampered with,
- agents execute at a sufficiently low level to provide direct filesystem access, and
- filesystem access is obtained immediately upon agent arrival so processes are unable to detect that a scan is imminent.

An existing security exposure is that communication must be decrypted at endpoints in order to be processed providing the potential for tamper with data upon agent arrival at a remote gateway. Communication between nodes or agents is handled by using SSL [24] for authentication and encryption as well as to prevent man-in-the-middle attacks [25]. Secure communication is ensured under SSL by strong cryptography. A vulnerability remains, however, in that encrypted data must be decrypted prior to use. Once it is decrypted, it becomes a potential avenue for tampering and is a future topic of research in secure microprocessor hardware [26].

In the current version of CONFIDANT, each agent maintains a list of available gateways including the location of each committee member. Prior to agent dispatch, a random number is generated to select a destination gateway. If the selected gateway is occupied, the agent will travel to the next available gateway. Future research includes modeling the network as a graph or Markov chain and performing statistical coverage testing to determine optimal agent transitions. Ongoing work with CONFIDANT including evaluation IDS response to insider tampering is presented in [27].

References

[1] J. McHugh, A. Christie, J. Allen, Defending yourself: The role of intrusion detection systems, IEEE Software 17 (5) (2000) 42–51.

[2] J. McHugh, A. Christie, J. Allen, Intrusion detection: Implementation and operational issues, Software Engineering Institute Computer Emergency Response Team White Paper (2001).
URL http://www.stsc.hill.af.mil/crosstalk/2001/jan/mchugh.asp

[3] S. Grafinkel, G. Spafford, A. Schwartz, Practical Unix & Internet Security, O'Reilly and Associates, 2003.

[4] K. Seifried, Linux administrator's security guide – attack detection.
URL http://gd.tuwien.ac.at/opsys/linux/lasg-www/attack-detection

[5] G. H. Kim, E. H. Spafford, Experiences with tripwire: Using integrity checkers for intrusion detection, Tech. Rep. CSD-TR-94-012, Department of Computer Sciences, Purdue University (1994).

[6] Tripwire, Inc., Tripwire for servers for security & network management.
URL http://www.tripwire.com/products/servers

[7] Tripwire, Inc., Tripwire manager for enterprise – wide security & network management.
URL http://www.tripwire.com/products/manager

[8] R. Lehti, Advanced intrusion detection environment.
URL http://www.cs.tut.fi/~rammer/aide.html

[9] Rocksoft, Veracity - nothing can change without you knowing: Data integrity assurance.
URL http://www.rocksoft.com/veracity/

[10] E. L. Cashin, Integrit file verification system.
URL http://integrit.sourceforge.net

[11] R. F. DeMara, A. J. Rocke, Mitigation of network tampering using dynamic dispatch of mobile agents, Computers & Security 23 (1) (2004) 31 – 42.

[12] M. Crosbie, G. Spafford, Active defense of a computer system using autonomous agents, Tech. Rep. 95-008, COAST Group, Department of Computer Sciences, Purdue University (February 1995).

[13] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, D. Zamboni, An architecture for intrusion detection using autonomous agents, Proceedings. 14th Annual Computer Security Applications Conference (1998) 13–24.

[14] M. C. Bernardes, E. dos Santos Moreira, Implementation of an intrusion detection system based on mobile agents, Proceedings. International Symposium on Software Engineering for Parallel and Distributed Systems (2000) 158–164.

[15] W. Jansen, P. Mell, T. Karygiannis, D. Marks, Applying mobile agents to intrusion detection and response, National Institute of Standards and Technology, Computer Security Division (1999).
URL http://csrc.nist.gov/publications/nistir/ ir6416.pdf

[16] C. Costa, TACH design concept, Lockeed Martin Corporation (1998).

[17] G. Wang, R. F. DeMara, A. J. Rocke, Mobility-enhanced file integrity analyzer for networked environments, in: Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics, and Informatics, Vol. 2, 2005, pp. 341 – 346.

[18] D. A. Frincke, D. Tobin, J. C. McConnell, J. Marconi, D. Polla, A framework for cooperative intrusion detection, in: Proceedings of the 21st NIST-NCSC National Information Systems Security Conference, 1998, pp. 361 – 373.
URL http://citeseer.nj.nec.com/frincke98framework.html

[19] G. B. White, E. A. Fisch, U. W. Pooch, Cooperating security managers: A peer-based intrusion detection system, IEEE Network (1996) 20–23.

[20] P. Kotzanikolaou, M. Burmester, V. Chrissikopoulos, Secure transactions with mobile agents in hostile environments, in: Proceedings of the 5th Australasian Conference on Information Security and Privacy, 2000, pp. 289 – 297.

[21] W. Jansen, Countermeasures for mobile agent security, Computer Communications, Special Issue on Advanced Security Techniques for Network Protection.

[22] W. Jansen, T. Karygiannis, NIST special publication 800-19 – mobile agent security, National Institute of Standards and Technology.
URL http://csrc.ncsl.nist.gov/mobilesecurity/Publications/ sp800-19.pdf

[23] CERIAS - autonomous agents for intrusion detection.
URL http://www.cerias.purdue.edu/about/history/coast/projects/ aafid.php

[24] OpenSSL: Open source toolkit implementing the secure socket layer protocol.
URL http://www.openssl.org/

[25] P. Burkholder, SSL man-in-the-middle attacks, Tech. rep., The SANS Institute (2002).
URL http://www.sans.org/rr/papers/60/480.pdf

[26] IBM PCI cryptographic coprocessor.
URL http://www-3.ibm.com/security/cryptocards/library.shtml

[27] A. J. Rocke, R. F. DeMara, Trusted detection of unauthorized filesystem modifications to combat insider tampering, Tech. Rep. UCF-ECE-0410, School of Electrical Engineering and Computer Science, University of Central Florida (November 2004).
URL http://netmoc.cpe.ucf.edu:8080/internal/yearReportsDetail.jsp?year=2004&id=0410