Mitigation of Network Tampering Using Dynamic Dispatch of Mobile Agents[†]

Ronald F. DeMara and Adam J. Rocke
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2450
demara@mail.ucf.edu arocke@pegasus.cc.ucf.edu

Abstract

Detection of malicious activity by *insiders*, people with legitimate access to resources and services, is particularly difficult in a network environment. In this paper, a novel classification of *tampering modes* is identified that can be undertaken by insiders against network Intrusion Detection Systems (IDSs). Five categories of tampering modes are defined as *spoofing*, *termination*, *sidetracking*, *alteration of internal data*, and *selective deception*. These are further distinguished specifically toward IDS sensor, control, and alarm categories such as *spoonfeeding*, *sugarcoating*, and *scapegoating*.

The *Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions*, or *CONFIDANT*, uses distributed mobile agents to mitigate these tampering exposures. CONFIDANT employs techniques such as *encapsulation*, *redundancy*, *scrambling*, and *mandatory obsolescence*. This paper describes how these mitigation techniques are applied within the CONFIDANT framework. The approach focuses on evaluating file integrity through the use of dynamically dispatched mobile agents.

Index Terms

Host-based security with network components, file integrity analyzers, insider risks, tampering modes, mobile agent behaviours

I. Introduction

The goal of an *Intrusion Detection System (IDS)* is to identify occurrences of security breaches capable of compromising the integrity of resources or services. *File Integrity Analyzers* are a class of related tools that automatically verify the content of security-critical files. Frequently referred to as *tripwires*, they attempt to detect if files have been modified in unauthorized ways. Once suspicious modifications are detected by triggering the tripwire, the analyzer may alert a security administrator or invoke some type of automated response. Alternatively, file analyzers can provide guidance for damage control, such as identifying the modified files needing to be restored or hooks installed by the attacker to facilitate subsequent access.

Over the past 20 years, numerous research-oriented [1] [2] [3] [4] [5] and publicly-available [6] [7] [8] [9] [10] frameworks have been developed to mitigate a variety of intrusion exposures. In practice, exposures range from the comparatively benign deployment of a Trojan horse that resets web browser home pages without permission, up through revenge by a distraught system administrator who malevolently alters enterprise-critical executables.

[†]Supported in-part by National Security Agency MDA904-99-C-2642 subcontract and a Lucent Florida Universities Fellowship.
[†]Publisher-formatted version appears in: Computers and Security, Volume 23, Issue 1, February 2004, Pages 31-42. doi:10.1016/S0167-4048(04)00068-9

A. Insider Tampering Exposures

A malicious action by a legitimate user, referred to as *insider tampering*, is particularly challenging to deal with. Insiders such as system administrators have broad access to sensitive resources, an extensive understanding of internal procedures, and frequent opportunities to carry out unauthorized use. Thus, attacks perpetrated by knowledgeable insiders have the potential to be more devastating than those that are externally-originated. Moreover, a common tactic of outsiders is to obtain limited access then elevate their privilege to that of an administrator with a high capability levels. For this reason, insider risk is recognized as an exposure where few useful tools exist and significant exposures receive relatively little attention [11] [12] [13]. Within the academic community, the insider problem is recognized as a difficult one. Neumann and Porras classify the detection of hitherto unknown attacks as very challenging open problems, citing subtle forms of misuse by insiders as a particular concern [5]. Recent DoD Workshops have identified the need for insider threat models as urgent [12]. In terms of a likely target of tampering within the domain of insiders, Axelsson identifies determination of the nature of attacks on the intrusion detection components as a fundamental unanswered question [14].

A conventional approach to mitigating insider tampering is to require concurrent login from 2 or more trusted individuals before granting administrator-level privileges. In practice, it is likely that one individual will leave the other unattended at some point in time after simultaneous login. Even if this policy is abided by completely, it is unreasonable to expect that two administrators with different backgrounds and familiarization levels to be fully cognizant of the rationale and impact of each action the other undertakes.

B. Motivation for Assessing File Integrity Frameworks

From a high level perspective, an IDS monitors actions in the computing environment in order to identify possible signs of an attack. Since file integrity analyzers may perform periodic inspections, some file integrity frameworks may not be considered full-fledged IDSs [15]. Still they occupy an important role as vital components of an intrusion detection environment. In this paper, the rationale for concentrating on the file integrity problem includes several compelling motives:

- cryptographically-hashed file integrity verification is by far the most popular tampering detection approach among installed systems,
- integrity verification approaches can positively detect many foreseen and even some unforeseen intrusions while maintaining a low false alarm rate,
- regardless of the initial intrusion pathway, a popular first action of intruders is to open a backdoor by altering system executables to facilitate future access [16],
- with possible exception of hardware modifications, persistent compromises to a network usually involve alteration of system executable or configuration files,
- integrity analyzers act as a damage control mechanism to locate altered data so they become second-line targets regardless of intrusion pathway, and
- rudimentary alterations can defeat a number of existing file integrity analyzers.

Tampering modes are identified as attacks undertaken to corrupt an intrusion detection framework. This paper presents a discussion of tampering modes present in network-based file integrity analyzers and introduces *CONFIDANT*, the *Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions*. In agreement with the meaning of the word *confidant* as “a most trusted servant”, it aims at trusted detection of unauthorized modification of executable, data, and configuration files.

In this paper, the tampering modes and mitigation techniques addressed by the CONFIDANT framework are enumerated. Section II discusses existing frameworks to identify broad classes of vulnerabilities. These are further formalized in Section III using an abstract model of IDS sensors, control mechanisms, and alarms. Section IV introduces CONFIDANT and addresses the defined vulnerabilities. Section V outlines future work.

II. Existing IDS and File Integrity Frameworks

Previous IDS and file integrity frameworks often relied on a client-server architecture to perform tasks ranging from file integrity analysis to user profiling. More recent agent-based approaches offer the potential of increased adaptability, reduced communication costs, and fault tolerance. The following discussion is restricted to various commercial and open source file analyzers and related multi-agent IDS frameworks. General purpose multi-agent IDS frameworks are investigated as few frameworks exist that focus on file integrity.

File integrity tools use one or more cryptographically-based hash mechanisms such as CRC32, SHA-1, or MD5 to compute a signature for monitored files [16] [17]. Signatures are subsequently recomputed and compared to previous values in order to detect if a file has been modified. A number of the most widely installed file integrity analyzers are reviewed in [18] and [19]. The relevant features of a few of them are discussed below.

A. Previous Conventional File Analyzers

While *Tripwire* [6] is the most popular commercial file integrity analyzer, other commercial and open source alternatives exist. Selected file analysis tools are listed in Table I. Tripwire utilizes a *policy file* to describe the expected behaviour of system and data files, identify files that are expected to change, and the types of changes permitted to each file. A *baseline database* is created using hash functions according to the policy file as reference to detect file modifications. In a networked environment, Tripwire on individual hosts can interact with a *Tripwire Manager* 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. Other file analyzers such as *AIDE* [9], *Veracity* [10], and *integrit* [20] operate similarly although AIDE aims toward at removing particular limitations in Tripwire and integrit focuses on essentials. Other tools exist that exhibit unique features. For instance, Nabou [21] can be used as a process monitor while SMART Watch [22] detects file system changes in near-real time by not using periodic timers.

TABLE I
Selected Conventional File Integrity Analyzers

IDS Framework Name	Availability	Execution Model	Comment
Tripwire	Commercial and Open Source	Client-Server	The most popular commercial file integrity analyzer
AIDE	Open Source	Single Host	Created as a response to the commercialization of Tripwire
integrit	Open Source	Single Host	Designed to be a simple alternative to Tripwire and AIDE
Veracity	Commercial	Client-Server	Concentrates on manipulating snapshots of directory trees
Nabou	Open Source	Single Host	Can be used as a process monitor
SMART Watch	Commercial	Single Host	Detects changes without relying on periodic timers

While some file analyzers have taken steps to reduce tampering exposures, mitigation of risks from knowledgeable insiders remains as an evolving area. For instance, AIDE product literature warns that its own integrity cannot be guaranteed as AIDE's binary and/or baseline database can also be altered. In fact, their website [9] warns that a hacked version is being maliciously distributed. One alternative recommended during Tripwire installation is to record the tool's binary files on write-once media. However, media within control of the system administrator becomes vulnerable to exchange. To address these concerns, tools such as Tripwire encrypt their baseline and verify its congruence with the policy file used to generate it. Tripwire also utilizes SSL communication protocols and triple DES encryption for critical transmissions. Yet, these

tools are still subject to tampering. A complete discussion of the modes by which tampering can occur are presented in Section III.

B. Previous Agent-based IDS and File Integrity Approaches

Recently agents have been proposed as a technology to overcome limitations in a variety of intrusion detection applications beyond file integrity. Rationale for considering agents in an IDS ranges from increased adaptability for new threats to reduced communication costs. Since agents are independently executing entities, there is the potential that new detection capabilities can be added without completely halting, rebuilding, and restarting the IDS. Other potential advantages are described by Jansen [23], and by Kruegel [24] who also identifies downside tradeoffs including increased design and operational complexity. Use of mobile code itself introduces new security exposures and mitigation methods as discussed in [25]. Although numerous agent-based IDS frameworks have been proposed [23] or constructed, few have aimed at addressing exposures from insider tampering.

1) Agent-based Anomaly Detection: The *Autonomous Agents for Intrusion Detection (AAFID)* framework [8] developed at Purdue University is a leading IDS project employing autonomous agents for data collection and analysis. AAFID utilizes *agents* hosted on network nodes, *filters* to extract pertinent data, *transceivers* to oversee agent operation, and *monitors* to receive reports from transceivers. These entities are organized into a hierarchical architecture with centralized control. While a transceiver may report to multiple monitors to provide redundancy, monitors as well as AAFID agents can remain at a single physical address upon deployment.

Cooperating Security Managers (CSMs) [1] enable individual distributed intrusion detection packages to cooperate in performing network intrusion detection without relying on centralized control. Each individual CSM detects malicious activity on the local host. When suspicious activity is detected, each CSM will report any noteworthy activity to the CSM on the host from which the connection originated. The local CSM will not notify all networked systems, but rather only the system immediately before it in the connection chain.

Other agent-based hierarchical architectures include the Intelligent Agents for Intrusion Detection project at Iowa State University [26]with a centralized data warehouse at the root, data cleaners at the leaves, and classifier agents in between.

Bernardes and Moreira have proposed a hybrid framework with partially distributed decision making under the control of a centralized agent manager [27]. Agents are deployed to observe behaviour of the system and users. Agents communicate via messages to advise peers when an action is considered suspect. The architecture is structured into four distinct layers. When an agent considers an activity to be suspect, an agent with a higher level of specialization for the suspected intrusion is activated. Agents then report their findings to a centralized manager.

In these systems, distribution of some aspects of the data collection and decision-making processes suggest important features to help diffuse distinct tampering points. Yet, whenever agents are managed from a common server, clearly identifiable targets for tampering remain. Furthermore, the use of one or more centralized repositories leave at least some portion of the network exposed to tampering and denial of service attacks. Even if an autonomous mobile decision-making agent was to detect a problem, interlocking mechanisms would be necessary to preclude any accidental or malicious removal, delay, or spoofing the agent.

2) Mobile Agents Supporting Remote-Access Detection of Misuse and Viruses: The University of Idaho has developed the *Hummingbird* framework [28] for managing misuse data. Hummingbird agents are neither autonomous nor mobile but do illustrate important methods to mitigate tampering such as including validated transactions between stationary decision-making centres, redundant data collectors, and use of Kerberos. The project and test cases used focus on cooperative intrusion detection if sharing of data is a viable option between distinct hosts across an enterprise.

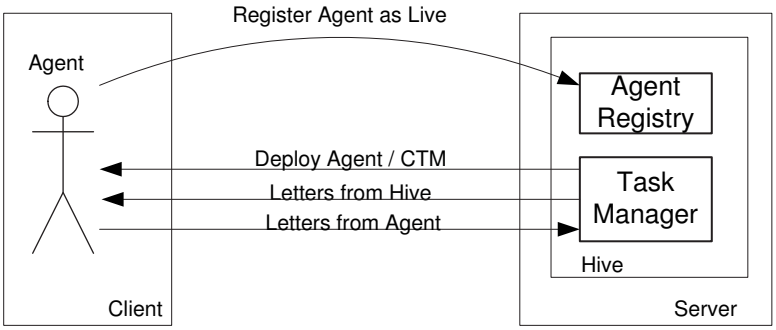

Fig. 1. TACH Architecture

The *Tethered Agent and Collective Hive (TACH)* concept for remote-access system management was defined by Costa [29] at Lockheed Martin in 1998. Figure 1 shows a high-level view of TACH. The three components of the architecture include a centralized *Hive* to keep track of agents and collected data, a *Task Manager (TM)* to assign priority codes and conditions of task execution, and an *Agent Registry (AR)* to track fingerprints of agents. Working with Lockheed, the University of Central Florida (UCF) designed and developed an Aglet-based [30] framework for TACH including mobile agents for virus detection [31] [32] and misuse detection [33]. Upon deployment by the TM, agents establish communication with the Hive and the AR registers the agent as "live" then executes tasks as defined by its *Customized Task Module (CTM)*. A CTM is a plug-in that allows a standardized TACH framework to be more readily customized to new agent behaviours without the need to significantly modify other components or the communication protocols between agents and the Hive. Limitations of TACH include the use of a centralized Hive for agent control and a period communication protocol between agent with time-out detection used to detect status changes in the agents. If the hive is disabled then the entire TACH system has been compromised.

3) File Verification: Based on experience with TACH at UCF, the *File Integrity using Cooperating Agents (FICA)* framework [34], a second-generation system, was developed using Concordia mobile agents, MD5 protocols, and Java security APIs. FICA deploys two agent behaviours, an *initiator* to travel to remote nodes to create a baseline and an *examiner* to compute new digests for comparison with the original baseline. As with other file integrity analyzers, FICA must rely upon write-once media for configuration files and immutable digests. Also, since a single centralized server for dispatching was implemented, serious exposures remain.

III. IDS Exposures and Tampering Modes

Intrusion exposures have been classified in the literature based on numerous features. These include the access pathway of the attacker, the system vulnerability exploited, and the procedures or data being targeted [35]. Classifications based on the intruder's intent have also been developed [36] as well as consideration of the intruder's knowledge level [37]. In this section, a widely-applicable classification is defined for exposures facing the IDS itself.

Attacks undertaken to corrupt an intrusion detection framework are identified as IDS *tampering modes*. Tampering modes signify critical vulnerabilities for several reasons. First, their targets are the IDS facilities themselves that must be relied upon to detect adversarial events. Second, protection for the entire system may be compromised if tampering is successful. Third, crucial exposures to insider risk exist because an IDS framework is under the direct control of administrators.

In the case of file analyzers, tampering may be directed at each IDS component. As identified in Table II, *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. Each of these subsystems is vulnerable to *spoofing*, *termination*, *sidetracking*, and *internal alterations* as described by the tampering modes identified below.

TABLE II
TAMPERING MODES UNDER CONSIDERATION

IDS Vulnerability		Tampering Mode	Instantiation for File Integrity Analysis
Spoofing data to:	Sensor	Spoonfeeding	Alternate data stream is conveyed during file scan
	Control	Sugarcoating	Unfavourable cryptographic digest is modified to appear as the desired result
	Alarm	Recanting	Fraudulent command is issued to deactivate alert
Termination of:	Sensor	Blindfolding	Detection mechanism is disabled
	Control	Commandeering	Decision-making process is usurped
	Alarm	Soundproofing	Notification mechanism is eliminated or muted
Sidetracking of:	Sensor	Blockading	Resource usage is forestalled to starve access
	Control	Pacing	Scan timing reference is corrupted or execution priority is overwhelmingly reduced
	Alarm	Scapegoating	Attention is diverted to a contrived distraction
Alter internal data in:	Sensor	Retroactive Baselining	Reference values for digests are modified
	Control	Descoping	Exemption is added to policy file to exclude scan coverage of unauthorized modifications
	Alarm	Value Jamming	Stand-alone process continuously writes `FALSE` into the memory location of status indicator
Selective deception		File Juggling	Target files interchanged before and after scanning

A. Spoofing-based Tampering

Spoofing attacks transmit counterfeit data to mislead the recipient. An IDS sensor is vulnerable to spoofing by *spoonfeeding* it information that is not present in the target file. Instead of accessing the file's real contents, an adversarial stream of data is provided in a compulsory manner. Data spoonfeeding can be realized at the kernel level by hacking I/O routines to omit or insert information. Alternatively, a file request from the IDS may be redirected to an unmodified copy. When implemented successfully, these modifications are disguised by including their changes among those items being masked. For example, consider when IDS or kernel files are loaded for execution. Initially suppose hacked versions get referenced. Later, the identical filenames may be presented for verification. However these executables might be hacked to redirect any such read requests to the original versions instead. The adversary's capacity to distinguish `execute` system calls from `read` calls enables spoonfeeding when necessary.

IDS control is vulnerable to *sugarcoating* of unfavourable reports before evaluation. For instance, suppose that unauthorized modifications render an incongruent digest for a target file. However, the reports conveyed to decision-making routines are altered to provide digests for the unmodified file instead. Even if all intra-IDS communication is encrypted and secure, an exposure still exists. Modified IDS routines might sugarcoat reports about target files as well as incongruities about themselves.

The alarm subsystem is vulnerable to spoofing by *recanting* alert notifications. For instance, a counterfeit notice to discontinue an alert might be sent ex-post-facto, as if the alert was only a false alarm. If the alert is recanted in a timely manner then the alarm may not be properly realized or go unperceived.

B. Termination-based Tampering

Although more conspicuous, outright termination of IDS mechanisms can facilitate potential modes of tampering. As listed in Table II, *blindfolding* attempts to exploit this vulnerability. By terminating a sensor, the perceptive ability of the IDS may decline. Without proper safeguards, an IDS adversary might attempt to disable local sensor processes to obscure unauthorized file changes.

Termination exposures also include the *commandeering* of IDS control. This refers to the overriding or bypassing of IDS decision-making operations. Commandeering of the IDS may involve removing critical

routines from an executable. Hence when unfavourable information is presented, the logic routines needed to detect intrusion are missing. For instance, suppose an IDS routine is stored on a host's hard drive in standard unencrypted executable format. It then becomes vulnerable to alteration using a disk sector editor. The critical opcode to bypass is the `Test_and_Branch` instruction selecting the 'intrusion detected' branch of the code. A discovery testbench that uses a divide-and-conquer approach to replace `Test_and_Branch` instructions and then observe the effect is not an implausible exposure. Sufficiently capable adversaries might also arrange for IDS modifications during loading or after the code resides in memory.

Termination-oriented vulnerabilities at the alarm level include *soundproofing* techniques. Soundproofing an IDS framework involves muting the alarm to preclude end-user notification. This includes terminating the alarm process or halting inclusion of culpable entries in a report or database. Likewise, an IDS adversary may also disable alarm capability via bypassing of execution. For example, suppose the first instruction in an alarm routine is replaced with a `Return_from_Subroutine` opcode. The appropriate location may be listed in a symbol table within the executable. It might also be reverse engineered from among the `Jump_Subroutine` destination addresses and known system calls.

C. Sidetracking-based Tampering

More sophisticated, yet less detectable, tampering modes attempt to sidetrack the IDS. They interfere with file integrity operations through collateral means, such as Denial of Service (DoS) attacks against IDS mechanisms. *Blockading* attempts to isolate a sensor from needed access to a target file or device. Some integrity frameworks can be blockaded by not relinquishing exclusive non-preemptive privileges. Robust file verification involves taking into account that blockading attacks are plausible against a wide range of IDS operations.

IDS control mechanisms are also vulnerable to sidetracking by *pacing* techniques. Pacing alters the execution rate in order to decrease the effectiveness of the IDS. A direct way to pace a file analyzer is by increasing the period between integrity scans. Similarly, an adversary may attempt to postpone scanning by resetting the time-of-day clock. Implicit means such as demoting IDS execution priority can also be problematic. Either approach might be sufficient to provide an intruder with an undetected window of access.

At the alarm level, *scapegoating* is an IDS tampering mode that focuses blame on unrelated events. For instance, an alarm may be intentionally triggered for a decoy cause in order to camouflage the actual attack. Similarly, a multiple alert DoS attack may be launched that overwhelms the alarm's processing mechanism or intended human recipient. Once either of these becomes overloaded, the system is potentially exposed to undetected intrusions.

In general, sidetracking-based tampering modes are difficult to mitigate. They exploit vulnerabilities whereby file analysis is impaired, yet each IDS component remains installed, unmodified, and in operation. This implies mitigation of IDS sidetracking will require ancillary techniques beyond the file verification methods themselves.

D. Internal Data Tampering

IDS data structures can also be vulnerable to tampering. *Retroactive baselining* is the after-the-fact modification of reference values. For example, integrity checkers create baseline files that store the digests for the initial state of files. Retroactive baselining corrupts these reference values. The updated baseline may reflect the digests after file modifications were made. Corresponding restoration of the baseline's access descriptors may help obscure the changes.

At the control level, an IDS is vulnerable to reductions in its operating scope to exclude analysis of malicious events. With respect to file analyzers, IDS *descoping* amounts to excluding integrity verification of unauthorized changes. The range of verification may be descoped by tampering with the analyzer's policy

```
   while(1) {
1:          sleep(95 × τ/100);
2:          rename(unmodified_version,original_filename);
3:          sleep(5 × τ/100);
4:          rename(modified_version, original_filename);
           }
```

Fig. 2. Fixed-Interval File Juggling Script.

file. Policy files are employed by integrity checkers to limit false positives. False positives correspond to file modifications that are routine, expected a-priori, and not indicative of any malicious intent. So this tampering mode can involve appending entries to the exception list in the policy file. Normally, policy files are maintained by the system administrator which introduces a straightforward insider pathway. Similarly, there is exposure to the analyzer being spoonfed the adversary's version of the policy file.

With respect to alarm mechanisms, an IDS may be vulnerable to *value jamming*. This tampering mode employs an independent adversarial process. This high-priority process continually writes FALSE to a status flag maintained in memory. Through repeated jamming of a status value, it may be feasible to preclude sustained establishment of an alert. This underscores need for proper access control to memory pageframes for robust IDS operation.

E. Selective Deception

Selective deception refers to tampering modes that are colloquially known as double-dealing. By way of analogy, consider an unscrupulous casino dealer who selectively issues playing cards from two decks to defraud the recipient. One problematic form that impacts file analyzers is *file juggling*. Realistically, file integrity verifiers can only inspect target files intermittently. So these tools are susceptible to the existence of modified data at times other than file verification.

Restoration of authentic data immediately prior to scanning might be achieved through a *file juggling script*. If an adversary is able to detect when a file integrity check will occur then a modified file can be temporarily reverted back to its original state. Suppose verification is set to occur with a pre-defined interval τ. The IDS may be susceptible to the file juggling script shown in Figure 2. Line 1 provides access to the modified file 95% of the time, Line 2 restores the unmodified version prior to checking, Line 3 waits for the integrity scan to complete, and Line 4 restores the modified version. More extensive operation would be necessary to restore the original file creation time and owner.

Alternatively, suppose that scans are scheduled at random intervals. Exposure to active detection of scanning operations would become a concern. In particular, the IDS adversary may replace Line 2 in Figure 2 with `(while not(exec("grep scan_process < top")))`. This creates a busy waiting loop until the `scan_process` begins executing at which time files are interchanged. Fine-tuning of the file juggling reaction time can be obtained by tweaking process priorities and resource blockades. In other words, a combination exposure exists from file juggling, pacing, and blockading tampering modes.

IV. CONFIDANT Operational Concept

Existing conventional and agent-based systems do not concentrate on mitigating insider tampering risks nor are their architectures easily adapted to handle them. CONFIDANT mitigates tampering vulnerabilities listed in Table II by employing a distributed control scheme realized with mobile agents. Distributing both data and control using agents is not only a key design consideration to mitigate insider tampering but also removes the single point of failure present in existing frameworks. By enabling agents to monitor the host file system and compare current and baseline file signature values, CONFIDANT is considered an agent-based signature-matching system with active network components. Also, since alarms are triggered by changes to monitored files, CONFIDANT signals intrusions in a behaviour-based manner [15].

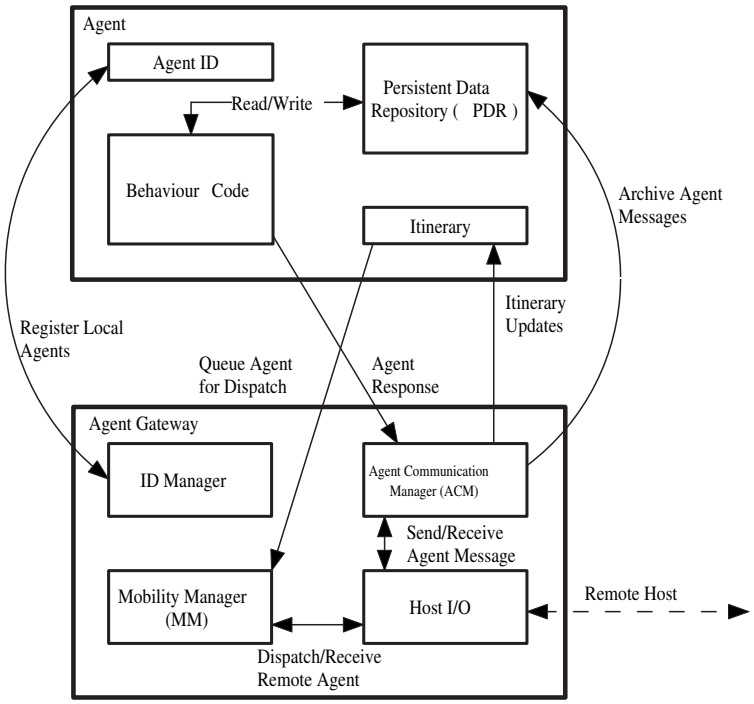

Fig. 3. General Agent Structure

CONFIDANT extends the TACH and FICA frameworks based upon the requirements listed in Table III. These three critical requirements involve the integrity of the agent execution, the inter-agent communication, and the processing results maintained during execution. CONFIDANT agents must be 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. With regard to these requirements, the listed assumptions are attainable and considered modest given the difficult challenges posed by insider risk. While an insider has full and direct access to any computer system resource, the stated assumptions coupled with use of mobile agents significantly diminishes the ability of any insider to *simultaneously* modify every agent in order to compromise file integrity capabilities.

The general structure of a CONFIDANT agent is shown in Figure 3. CONFIDANT agents contain executable *Behaviour 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 realtime 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.

TABLE III
CONFIDANT OPERATION ASSUMPTIONS

Requirement	Operating Assumption
Agents execute in a protected environment.	Initial configuration is well-formed and completely installed.
Agent interactions are robust.	Agent transport and communication occur via SSL to preclude spoofing.
File information obtained by agents is legitimate.	Agents have direct disk access to ensure accurate file validation.

TABLE IV
COMPARISON OF REPRESENTATIVE FILE INTEGRITY AND AGENT-BASED IDS APPROACHES

File Integrity Analyzer / IDS	Execution Model	Agent Form	Agent-to-Agent Interaction	Single Point of Failure	Safeguards Against Insider Tampering
Tripwire, AIDE, Veracity, integrit	Client-Server	N/A	N/A	Yes	No
AAFID	Deployed Agent	Key-Value Pair	None	Yes	No
Bernardes-Moreira	Mobile Agent	Aglet	Create, Halt	Yes	No
TACH	Deployed Agent	Aglet	None	Yes	No
FICA	Deployed Agent	Concordia Java Object	Communicate	Yes	No
CONFIDANT	Mobile Agent	Concordia Java Object	Communicate, Create, Halt	No	Yes

A. Towards Insider-Robust Capabilities

Table IV shows a comparison of related IDSs and illustrates how the CONFIDANT framework differs from existing models. For instance, CONFIDANT uses mobile agents that communicate with each other. The Bernardes-Moreira model uses mobile agents but there is no communication between agents. Several other agent based models dispatch agents but those agents do not traverse the network domain.

All of the IDS frameworks described in Section II have limitations with respect to insider tampering and exhibit a single point of failure. Tripwire can be compromised by interfering with the database manager. Compromising a transceiver in AAFID will remove a portion of the network from the control of the IDS. TACH, FICA, and the Bernardes-Moreira framework can be defeated by compromising the agent server. The mobile agent approach in CONFIDANT provides an alternative to these labor-intensive protocols to mitigate some specific vulnerabilities. A mitigation strategy developed in CONFIDANT is identified for each tampering mode described in Section III is listed in Table V.

TABLE V
MITIGATION APPROACHES TO TAMPERING MODES DEVELOPED IN CONFIDANT

Tampering Mode	Mitigation Approach	Description
Spoonfeeding	Encapsulation	Vulnerable File I/O contained inside agent
Sugarcoating	Validated transactions	SSL used for messaging and transport
Recanting	Interlocks, scrambling	Agent transactions are interlocked and spatially distributed
Blindfolding	Redundancy, vulnerability seeding	Known exceptions are intentionally inserted to test detection status
Comandeering	Interlocks, scrambling	Agent interactions are interlocked, and spatially and temporally distributed
Soundproofing	Redundancy, interlocks	Alarms at each node with interlocked I/O
Blockading	Pulse-taking	Interlocked file bandwidth monitoring and alert mechanism
Pacing	Pulse-taking	Interlocked CPU throughput monitoring and alert mechanism
Scapegoating	Redundancy	Concurrent tracking via multiple agents
Re-baselining	Distinct Inception	Data dispatched with agent upon configuration
Descoping	Mandatory Obsolescence	Configure only upon initial startup then destroy configuration agents
Value Jamming	Redundancy, scrambling	Agent execution is spatially and temporally scrambled
File Juggling	Redundancy, scrambling	Unpredictable redundant scan scheduling

B. Mitigation of Spoofing-based Tampering

Spoofing occurs when counterfeit data is transmitted to the recipient. Three spoofing attacks are considered. The first is *spoonfeeding* a sensor information that is not present in the target file. As listed in

Table V this is mitigated in CONFIDANT by encapsulation of the interface between the agents and native services on the host. The *Agent Gateway* enable the agents to access the host filesystem directly.

Sugarcoating of unfavourable reports is mitigated in CONFIDANT by using SSL encryption for messaging and transport in order to validate all agent communication and transfer. Agents will also perform integrity verification on the agent gateway to determine if tampering has occurred at the gateway level. CONFIDANT mitigates *recanting* alert notifications by enabling transaction interlocking between agents. Agents must remain in constant communication. If agent communication is interrupted and handshaking is not established, the receiving agent declares that a suspicious activity has occurred and activates the appropriate alert.

C. Mitigation of Termination-based Tampering

Disabling an IDS sensor is called *Blindfolding*. This is mitigated in CONFIDANT by enabling multiple agents to perform similar tasks. Agents remain in constant communication so that if an agent is determined to be missing by not maintaining an appropriate communication channel, or if an agent gateway can not be contacted, an alert is initiated. Furthermore, one agent can alter a file in order to verify that other cooperating agents are correctly identifying file modifications.

Overriding of IDS decision-making operations, or *commandeering*, is mitigated in CONFIDANT by distributing all decision-making responsibilities in the form of redundant mobile agents. Agent transactions are interlocked and are both spatially and temporally distributed.

Soundproofing an IDS framework involves muting the alarm to preclude end-user notification. This is mitigated in CONFIDANT by providing communication with multiple agents and by interlocking I/O via the agent gateway. If messages from a remote agent is expected and not received, an alert is initiated. The gateway provides agents with direct access to system resources so that when an agent receives an alert notification, the alert can be reliably transmitted to the host.

D. Mitigation of Sidetracking-based Tampering

Some frameworks are subject to *blockading*, or isolating a sensor from needed access to a computer system component. CONFIDANT mitigates blockading by distributing the investigating and decision-making responsibilities. If agent throughput is limited and agents are not able to access either a network node or a service on the host within a specified time, the resource under investigation or the entire node is considered suspect and an alert is initiated.

Altering execution rates, or *pacing*, is mitigated in CONFIDANT by redundancy of agents. Multiple agents traverse the network to analyze files on remote computer systems. Each agent has a unique itinerary and work in conjunction with others. Agent actions are based on internal timers defined by individual agents and not on the time of day. An agent must provide status messages to cooperating agents in a timely manner or tampering is suspected.

An example of *scapegoating* is triggering an alarm as a decoy in order to hide an actual attack. This is mitigated in CONFIDANT by enabling multiple agents to pursue each simultaneous alert independently so that multiple alerts can be processed concurrently.

E. Mitigation of Internal Data Tampering

File integrity tools create an initial baseline reference for future file verification. *Retroactive baselining* modifies the reference values thus corrupting the baseline. This is mitigated in CONFIDANT by maintaining baseline data within each agent responsible for file integrity verification. When an agent computes a cryptographic digest for a file, the result is compared to internal baseline data encapsulated within multiple mobile agents. External data is not used as a baseline. If the internal data is modified, agent redundancy enables file verification to be performed by other agents.

IDS control components are subject to *descoping* by tampering with the initial policy configuration data. This is mitigated in CONFIDANT by including policy information within each agent. The list of monitored files is defined prior to initial agent dispatch and is not modified during network traversal. If the policy information for a critical file is somehow maliciously altered, redundancy ensures that particular file will be inspected by other agents.

Value jamming occurs at the alarm level and involves interference with a malicious high-priority process altering the contents of memory. This is mitigated in CONFIDANT by enforcing each agent to be responsible for maintaining file status information. It is possible to have multiple agents on a node at any given time so there is no single memory location that serves as a status flag. Memory locations used for status information can also vary each time an agent visits a node due to the agent itself occupying a different memory location. Since status flag memory locations can be different for each agent visitation, CONFIDANT is less vulnerable to tampering by jamming.

F. Mitigation of Selective Deception

In order for a framework to be subject to tampering by *file juggling*, an adversary must be able to predict when file integrity checks will occur in order to perform undetected file system modifications. Selective deception is mitigated in CONFIDANT by enabling multiple redundant agents each with a unique itinerary and scheduling parameters. Agent visitation does not occur at regular intervals. It is not required for an individual CONFIDANT agent to visit every node but coverage of all nodes is guaranteed by the use of multiple agents each with an independent itinerary.

V. Conclusion and Future Work

IDS tampering modes can be divided into five broad categories defined as spoofing, termination, sidetracking, altering internal data, and selective deception. These categories can be further identified as tampering directed specifically toward IDS sensor, control, and alarm categories. File-oriented tampering involves unauthorized modifications to permanent records maintained by the IDS including File Juggling, Retroactive Baselining, and Descoping. Some other tampering modes are process-oriented such as Value Jamming by modifying the contents of memory. Communication-oriented tampering includes Spoonfeeding and Sugarcoating which intentionally supply inaccurate data to IDS components. Some tampering modes are reactive to specific IDS operations such as Recanting of an assumed alarm while others are proactive such as Blindfolding, Commandeering, and Soundproofing that disable or mute IDS components and Blockading to cause starvation. Furthermore, IDS processes can also be tampered by Scapegoating and Pacing methods which modify the context in which the IDS operates.

Given a modest set of assumptions, tampering exposures can be reduced using the mitigation techniques such as those employed by CONFIDANT. These include encapsulation, validated transactions, interlocking, scrambling, redundancy, pulse-taking, distinct inception, and mandatory obsolescence. Although the distributed agent architecture increases design complexity, it also facilitates a much wider range of mitigation techniques.

Initial evaluation of CONFIDANT with regard to metrics of robustness against tampering, run-time performance, maintainability, and scalability are presented in a companion paper [38]. An important result is that since CONFIDANT focuses on detection of insider tampering, its testing need not emphasize specific routes whereby intrusive access was obtained. Future work includes developing additional sensor functionality beyond file integrity analysis.

References

[1] G. B. White, E. A. Fisch, and U. W. Pooch, “Cooperating security managers: A peer-based intrusion detection system,” *IEEE Network*, pp. 20–23, Jan/Feb 1996.

[2] D. Schnackenberg, K. Djahandari, and D. Sterne, “Infrastructure for intrusion detection and response,” *DARPA Information Survivability Conference and Exposition, 2000. DISCEX ’00*, vol. 2, pp. 3–11, 1999.

[3] A. J. Hoglund, K. Hatonen, and A. S. Sovari, "A computer host-based user anomaly detection system using the self-organizing map," *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000. IJCNN 2000*, vol. 5, pp. 411–416, 2000.

[4] R. Heady, G. Luger, A. Maccabe, and M. Servilla, "The architecture of a network level intrusion detection system," Master's thesis, Department of Computer Science, University of New Mexico, August 1990.

[5] P. G. Neumann and P. A. Porras, "Experience with EMERALD to date," in *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, April 1999, pp. 73–80. [Online]. Available: http://citeseer.nj.nec.com/neumann99experience.html

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

[7] Y. Fyodor, "'SNORTNET' - a distributed intrusion detection system," June 2000. [Online]. Available: http://snortnet.scorpions.net/snortnet.pdf

[8] E. H. Spafford and D. Zamboni, "Intrusion detection using autonomous agents," *Computer Networks*, vol. 34, no. 4, pp. 547–570, 2000.

[9] R. Lehti, "Advanced intrusion detection environment." [Online]. Available: http://www.cs.tut.fi/˜rammer/aide.html

[10] Rocksoft, "Veracity - nothing can change without you knowing: Data integrity assurance." [Online]. Available: http://www.rocksoft.com/veracity/

[11] P. Galiasso, O. Bremer, J. Hale, S. Shenoi, D. Ferraiolo, and V. Hu, "Policy meditation for multi-enterprise environments," *Computer Security Applications Conference, 1999. (ACSAC '99) Proceedings*, pp. 219–228, 1999.

[12] R. Anderson, T. Bozek, T. Logstaff, W. Meitzler, M. Skroch, and K. V. Wyk, "Research on mitigating the insider threat to information systems," *Workshop Proceedings, RAND Corporation Report CF-163*, August 2000.

[13] C. Kahn, "Tolerating penetrations and insider attacks by requiring independent corroboration," in *Proceedings of the 1998 Workshop on New Security Paradigms*. ACM Press, 1998, pp. 122–133.

[14] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[15] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, pp. 805–822, 1999.

[16] J. Rauch, "Basic file integrity checking," SecurityFocus.com, August 2000. [Online]. Available: http://www.securityfocus.com/focus/linux/articles/ fileinteg.html

[17] "Intrusion detection faq: What is the role of a file integrity checker like tripwire in intrusion detection?" SANS Institute, 2000. [Online]. Available: http://www.sans.org/newlook/resources/IDFAQ/integrity_checker.htm

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

[19] "File integrity checkers." [Online]. Available: http://www.networkintrusion.co.uk/integrity.htm

[20] E. L. Cashin, "Integrit file verification system." [Online]. Available: http://integrit.sourceforge.net

[21] T. Linden, "Nabou system integrity monitor." [Online]. Available: http://www.daemon.de/Nabou

[22] WetStone Technologies, Inc., "SMART Watch." [Online]. Available: http://www.wetstonetech.com/smartwatch_ns.html

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

[24] C. Kruegel and T. Toth, "Applying mobile agent technology to intrusion detection," in *ICSE Workshop on Software Engineering and Mobility*, 2001. [Online]. Available: http://citeseer.nj.nec.com/kr01applying.html

[25] Y. Kun, G. Xin, and L. Dayou, "Security in mobile agent system: Problems and approaches," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 1, pp. 21–28, 2000.

[26] G. Helmer, J. Wong, V. Honavar, and L. Miller, "Intelligent agents for intrusion detection," pp. 121–124, September 1998. [Online]. Available: http://citeseer.nj.nec.com/helmer98intelligent.html

[27] M. C. Bernardes and 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*, pp. 158–164, 2000.

[28] D. A. Frincke, D. Tobin, J. C. McConnell, J. Marconi, and D. Polla, "A framework for cooperative intrusion detection," in *Proc. 21st NIST-NCSC National Information Systems Security Conference*, 1998, pp. 361–373. [Online]. Available: http://citeseer.nj.nec.com/frincke98framework.html

[29] C. Costa, "TACH design concept," Lockeed Martin Corporation, 1998.

[30] IBM Research, "Aglets." [Online]. Available: http://www.trl.ibm.com/aglets

[31] J. Lu, "Mobile agent protocols for distributed detection of network intrusions," Master's thesis, University of Central Florida, 2000.

[32] Y. Zhu, "Decentralized control schemes for coordinating distributed processing activities of mobile software agents," Master's thesis, University of Central Florida, 2000.

[33] B. Kapoor, "Remote misuse detection system using mobile agents and relational database query techniques," Master's thesis, University of Central Florida, 2000.

[34] G. Wang, "Moible agent file integrity analyzer," Master's thesis, Department of Electrical and Computer Engineering, University of Central Florida, 2001.

[35] J. D. Howard and T. A. Longstaff, “A common language for computer security incidents,” Sandia National Laboratories, Sandia Report SAND98-8667, October 1998.
[36] T. Bott, “Evaluating the risk of industrial espionage,” in *Proceedings of the Reliability and Maintainability Symposium*, 1999, pp. 230–237.
[37] R. Anderson and M. Kuhn, “Tamper Resistance - a Cautionary Note,” in *Proceedings of the Second Usenix Workshop on Electronic Commerce*, November 1996, pp. 1–11. [Online]. Available: http://citeseer.nj.nec.com/anderson96tamper.html
[38] R. F. DeMara and A. J. Rocke, “Confidant operational concept,” School of Electrical Engineering and Computer Science, University of Central Florida, Tech. Rep. TR-RFD2003-02, 2003.