Mitigation of Insider Risks Using Distributed Agent Detection, Filtering, and Signaling*

Adam J. Rocke and Ronald F. DeMara
(Corresponding author: Adam J. Rocke)

Department of Electrical and Computer Engineering, University of Central Florida
Orlando, FL 32816–2450, USA

(Received Aug. 2, 2005; revised and accepted Sep. 10, 2005)

Abstract

An insider-robust approach to file integrity verification is developed using interacting strata of mobile agents. Previous approaches relied upon monolithic architectures, or more recently, agent frameworks using a centralized control mechanism or common reporting repository. However, any such distinct tampering-point introduces vulnerabilities, especially from knowledgeable insiders capable of abusing security-critical resources. In the Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions (CONFIDANT), the mechanisms for tampering detection, decision-making, and alert signaling are distributed and corroborated by autonomous agents. In this paper, the CONFIDANT file integrity verification framework is presented focusing on insider defense aspects. User capability classes are defined and critical physical tampering points in intrusion detection architectures are identified. CONFIDANT mitigation techniques of insider tampering exposures and example scenarios are presented.

Keywords: File systems management, multiagent systems, network-level security and protection, security kernels, user profiles and alert services

1 Introduction

Intrusion Detection Systems (IDSs) aim to accurately identify computer system attacks. Intrusion detection does not seek to provide a comprehensive defense, but does play a significant role in overall network security to identify exploitation of vulnerabilities [6, 14, 15, 20]. *File Integrity Analyzers* are not considered a full-fledged IDSs but are an important component of an intrusion detection environment [3]. They serve as a vital IDS component by performing filesystem inspections to detect suspicious modification to security-critical files [9, 10, 12, 19, 21, 22]. Identification of an unauthorized modification results in an alarm to a security administrator and also helps to identify software changes that might facilitate subsequent access to the system [5, 8].

1.1 Insider Tampering Risk

Malicious actions by a user with administrative-level access permissions, referred to as *insider tampering*, presents a significant challenge to existing file integrity analyzers. Insiders possess both the ability and opportunity to perform unauthorized computer system use. Thus, tampering by insiders is potentially more damaging than by those without administrator-level permissions. Identification of misuse by insiders, particularly attacks on intrusion detection components, is a challenging problem [1, 2, 11, 13, 16]. While some file analyzers have taken steps to reduce *tampering exposures*, avenues by which malicious activities are able to occur, eliminating or reducing the risk from knowledgeable insiders remains an evolving area of research.

1.2 CONFIDANT Objectives

The *Collaborative Object Notification Framework for Insider Defense using Autonomous Network Transactions (CONFIDANT)* aims at trusted detection of unauthorized modifications to filesystem data. The design of CONFIDANT is based on two goals in order to limit exposures present in existing frameworks reviewed in [4]. *Goal-1* is to reduce single point-of-failure exposures in existing IDS frameworks. Increasing barriers against insider tampering is *Goal-2*. These goals are addressed by:

1) identifying single point-of-failure exposures in IDSs to address Goal-1,

2) developing a taxonomy of insider risks to address Goal-2,

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

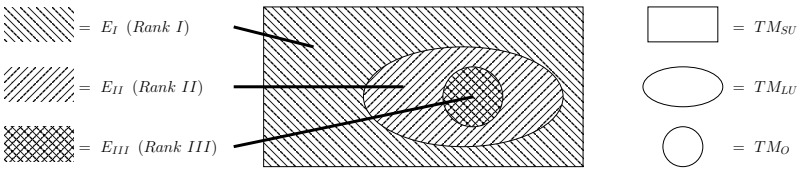

Figure 1: Relative rank of tampering modes addressed by CONFIDANT

3) designing a framework to address single point-of-failure and insider tampering exposures,

4) perform experiments to evaluate performance against both Goal-1 and Goal-2, and

5) evaluate performance of the proposed and existing approaches using comparative metrics.

In this paper, the CONFIDANT file integrity verification framework is presented focusing on insider defense aspects. IDS user capability and architectural tampering points are identified in Section 2. An overview of the CONFIDANT agent framework defined in [17] is provided in Section 3. Insider tampering modes and CONFIDANT mitigation techniques are provided in Section 4. CONFIDANT evaluation methodology and results are presented in [17] for Goal-1 and in [18] for Goal-2.

2 Mitigation of Insider Risks in Networked Environments

To attain a robust file integrity framework, it is necessary and sufficient to address vulnerabilities within the domain of administrators or *superusers*. Let TM_{SU} denote the set of *tampering modes* available to *superusers*, let TM_{LU} denote tampering modes of *local users* without superuser capabilities, and let TM_O denote tampering modes of *outsiders*. By definition, $TM_{SU} \supseteq TM_{LU}$ since superusers can perform all operations available to any other local user. Likewise, $TM_{LU} \supseteq TM_O$ since being a local user does not preclude conducting tampering activities available to outsiders. By transitivity, $TM_{SU} \supseteq TM_O$, hence tampering modes of superusers subsume vulnerabilities of both authorized local users and unauthorized outsiders.

Motivated by this subsumption relationship, a design flow for IDS frameworks is obtained that focuses on insider risks, rather than considering them as an afterthought. As depicted in Figure 1, *Rank I* exposures denote only those tampering modes available exclusively to superusers; *Rank II* exposures denote any non-Rank I modes available to local users, but not to outsiders; and *Rank III* modes denote any exposure that is not a Rank I nor Rank II exposure. More formally, let the set of exclusive exposures of *Rank x* be denoted by E_x. Thus, $E_I = TM_{SU} - TM_{LU}$, $E_{II} = T_{LU} - TM_O$, and $E_{III} = TM_O$. Based on these rankings, the required agent behaviors can be developed as follows:

1) identify E_I vulnerabilities,

2) postulate an IDS design against which all identified vulnerabilities are evaluated,

3) repeat Step 2 until a design is obtained capable of detecting all identified vulnerabilities,

4) identify E_{II} vulnerabilities,

5) verify the postulated design *already* meets all E_{II} vulnerabilities, or if it does not then return to Step 2,

6) identify E_{III} vulnerabilities, and

7) verify the postulated design *already* meets all E_{III} vulnerabilities, or if it does not then return to Step 2.

Using this design flow, many agent behaviors that were developed to detect only Rank I vulnerabilities were also capable of detecting Rank II and Rank III exposures. By focusing the agent development effort on superuser exposures, other vulnerabilities can be mitigated without the need to explicitly design mechanisms to address the lower rank exposures.

The tampering capability classes describe categories of individuals based on access to computing resources. Physical tampering points in a computer system architecture along with the most general tampering capability at each point are listed in Table 1 and illustrated in Figure 2. Tampering by outsiders, denoted TM_O, without physical access to computing resources can only be performed remotely. Thus, TM_O is limited to tampering with network resources. In addition to the E_{III} exposure of tampering over the network, local users are able to exploit E_{II} vulnerabilities including modifying local filesystem contents and memory locations based on permissions assigned by the administrator. For this reason, tampering modes of local users, denoted TM_{LU}, include points TP_{FS}, TP_{PT}, TP_{IC}, TP_{ID} in Figure 2. These stand for tampering points at the filesystem, process table, IDS code, and IDS data, respectively. Insiders have few restrictions on computer resource use. Administrators, unlike the other capability classes, can tamper with any filesystem resource or memory location as well as modify the system clock, shown as TP_{SC}.

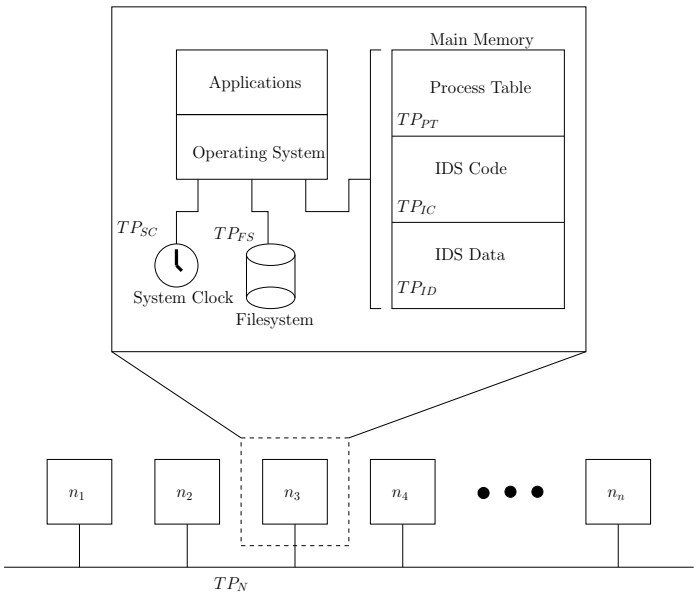

Figure 2: Physical architecture

3 CONFIDANT Agent Framework

The CONFIDANT mobile agent framework, defined in [17], consists of an *agent gateway* on each monitored host, four agent *behaviors*, and agent *interaction* operating in three *echelons*. The lowest echelon, the *Sensor level*, is responsible for *surveillance*. The middle echelon is the *Control level* which provides updates for agent itineraries as well as performs result collection and correlation. The uppermost echelon, the *Response level*, provides *alarm* notification.

3.1 CONFIDANT Terminology

Agents interact within *committees*. 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. All agents within a single committee correlate processing results. The set of all m committees within a monitored network is defined as $C = \{C^1, \ldots, C^m\} = \bigcup_{i \in m} C^i$. Multiple adjacent committees can be defined with each limited to a subset of monitored hosts in order to enhance scalability and adapt to physical network layout. Clustering and scalability of mobile agent based IDSs is discussed in [7] and [13]. Due to the overlapping nature of adjacent committees, $\bigcap_{i \in m} C^i \neq \emptyset$.

The *agent gateway* provides the interface between the agents and services on each host as well as the communication mechanism for the agents to travel over the network. Each gateway, $G = \{g_1, \ldots, g_k\}$, corresponds to a monitored network node, where k is the number of monitored hosts. Multiple agents within a single committee will attempt to travel to a gateway that is not currently hosting other members of the committee. If agents within a committee are able to congregate at a single gateway, a single point-of-failure exposure is created. The maximum number of agents that each gateway can host at one time is the subject of future research.

3.2 CONFIDANT Operational Assumptions

Critical requirements that exist for proper CONFIDANT operation include the integrity of agent operation, inter-agent communication, and processing results obtained. First, the initial configuration must be well-formed and completely installed prior to live network operation. Also, CONFIDANT reconfigurability following initial agent deployment is neither enabled nor desirable as configuration and management routines can facilitate insider tampering.

CONFIDANT agents verify the correctness of individual hosts on the network as well as other agents during operation. File contents and digest computation obtained by agents is trusted as agents have direct disk access to ensure accurate file validation. Agent interactions are robust as transport and communication occur via SSL to preclude spoofing. 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 compromise file integrity capabilities.

Even with assurance of an initial known safe state, host and communication correctness, and encrypted messaging, network hardware and the operating system kernels must remain free of tampering as CONFIDANT operates at the application-layer. Hardware and operating system tampering may subvert file integrity verification by introducing kernel trojans, corrupting agent data prior to SSL

Table 1: Computer system resource tampering points

Tampering Point	Definition	Associated Capability Class	
		User Apps	IDS Apps
TP_{FS}	Alteration of filesystem contents	TM_{LU}	TM_{SU}
TP_{PT}	Modifying the process table in memory	TM_{LU}	TM_{SU}
TP_{IC}	Changing application code while in memory	TM_{LU}	TM_{SU}
TP_{ID}	Changing application data while in memory	TM_{LU}	TM_{SU}
TP_N	Tampering from remote network nodes	TM_O, TM_{LU}, TM_{SU}	
TP_{SC}	Modification of the system clock	TM_{SU}	

ciphering, or modifying hardware drivers to redirect file access requests. Future research in techniques to mitigate tampering at the hardware and operating system level include bypassing the kernel to achieve direct device to device communication.

3.3 CONFIDANT Design

The CONFIDANT framework emphasizes mitigation of insider tampering vulnerabilities using a distributed control scheme realized with mobile agents. By distributing both control and data in the form of mobile agents across the network, insider tampering risks are mitigated and single point-of-failure exposures present in existing frameworks are eliminated.

CONFIDANT performs filesystem scans using the agent dataflow sequence illustrated in Figure 3. Integrity scans are performed by obtaining file contents, computing the file MD5 hash value, and comparing that result to the internal baseline data. Upon scan completion, the result is sent to other committee members to corroborate the result. Next, the agent travel operation commences. Prior to dispatch, the agent will parse the internal list of monitored gateways. Once a gateway is selected, the agent will send a travel request to committee members in order to maintain communication upon arrival at the remote gateway. When the agent arrives at the destination gateway, arrival notification is sent to committee members and filesystem scans resume.

Each agent maintains three internal timers. These timers define a waiting period for operation confirmation. If a message from a remote agent or gateway is expected, Δt_r defines the maximum time allowed for the message to be received prior to an alarm being generated. Also, Δt_s is the waiting period from when a message is sent until confirmation is received. The maximum time allowed prior to alarm notification for an agent to be dispatched to a remote gateway and send arrival notification to committee members is Δt_d.

Pseudocode for the scan operation is provided in Figure 4. The `computeMD5` function is performed to obtain the MD5 hash value of the monitored file. A `MD5_OK` event is created if the result matches the baseline. If a file modification is detected, a `MD5_Error` event is created. The event generated as a result of the scan is then distributed to other committee members. The name of each committee agent is stored locally in an array `ca[1..n]` along with the name of the gateway on which the agent is operating. A `foreach` loop is used to send the scan result to each committee member. The agent will then `sleep` for Δt_s to allow committee member responses to arrive. Responses are handled asynchronously and stored in an array. The responses are then processed to ensure the message was successfully conveyed.

In order to maintain agent interlocking, three communication sequences occur between distributed committee members as illustrated in Figure 5. The first sequence occurs upon file scan completion. Once the scan is complete, results are sent to committee members. The remaining sequences enable committee agents to maintain robust communication while travelling between remote gateways. Prior to dispatch, an agent will select an available gateway and notify committee members of intent to travel. Committee members respond in order to confirm that communication will be maintained upon arrival at the destination gateway. Once confirmation is received, dispatch commences. If the remote gateway is unavailable, an alarm is triggered, a new destination gateway is selected, and the interlocking communication process repeats. Upon arrival at the remote gateway, arrival notification is sent to committee members in order to ensure that future messages are transmitted to the correct gateway. An alarm may result from any of these three sequence if message acknowledgment is not received within Δt_s or if agent dispatch is not successful within Δt_d.

Pseudocode for the agent dispatch communication sequence is provided in Figure 6. First, `selectGateway` is called to determine the agent dispatch destination. A new `AgentTravelRequest` event is created and sent to committee agents as described previously. The notation `ca[i].g` represents the gateway on which the agent `ca[i]` is operating. A value of `null` indicates that travel or communication for the agent `ca[i]` has failed. Once responses are processed, the agent is dispatched to the remote gateway. Upon arrival at the destination, `sendArrivalNotification` is called to inform committee agents travel success.

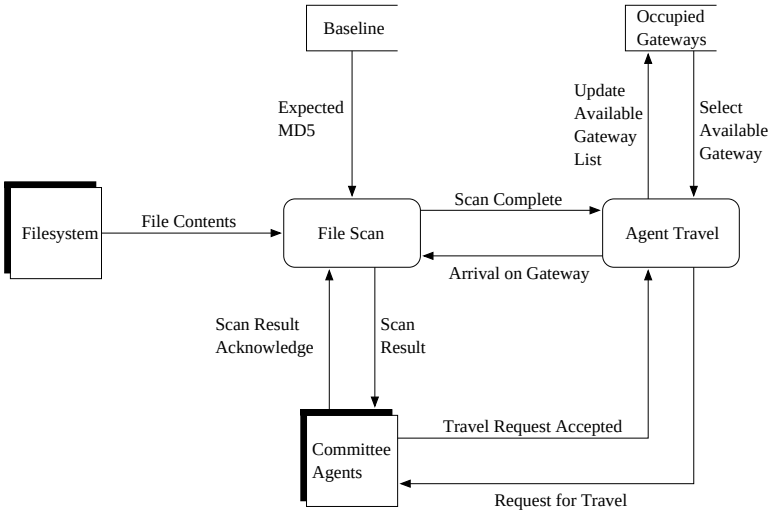

Figure 3: CONFIDANT dataflow diagram

```
gateways = g[1..m];
committee agents = ca[1..n];
responses = r[1..n];
scanresult.setValue(computeMD5(filename));
if (scanresult.getValue() == baseline) {
     result = new ConfidantEvent(MD5\_OK,
                                 scanresult.getValue());
} else {
     result = new ConfidantEvent(MD5\_Error,
                                 scanresult.getValue());
}
sendScanResult(result) {
     foreach i in ca[] {
          if (ca[i].g != null) {
          sendmsg(ca[i].g, result)
          }
     }
     sleep(delta_ts);
     processResponse();
}
```

Figure 4: CONFIDANT pseudocode for file scan operation

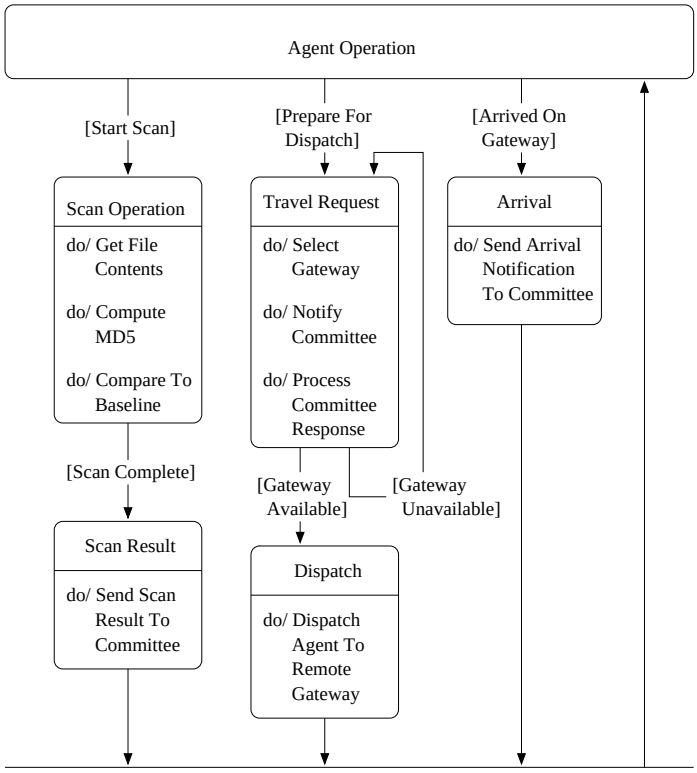

Figure 5: CONFIDANT operation state diagram

4 Insider Tampering Modes and CONFIDANT Mitigation Techniques

IDS tampering modes are defined in [4]. Each logical IDS subsystem defined previously is vulnerable to *Spoofing*, *Termination*, *Sidetracking*, *Altering Internal Data*, and *Selective Deception* as described by the tampering modes summarized below. The mitigation strategy developed for CONFIDANT for each tampering mode is listed in Table 2.

4.1 Spoofing-based Tampering

Spoofing occurs when counterfeit data is transmitted to the recipient. Three spoofing attacks are considered. The first is *Spoonfeeding* sensor information at TP_{FS} that is not present in the target file. As listed in Table 2 this is

```
gateways = g[1..n];
committee agents = ca[1..n];
responses = r[1..n];

availablegateway = selectGateway();
destination = new ConfidantEvent(AgentTravelRequest,
                                              availablegateway);
sendTravelRequest(destination) {
     foreach i in ca[] {
          if (ca[i].g != null) {
          sendmsg(ca[i].g, destination)
          }
     }
     sleep(delta_ts);
     processResponse();
     dispatch(destination);
     sendArrivalNotification();
}
```

Figure 6: CONFIDANT Pseudocode for Agent Dispatch

mitigated in CONFIDANT by encapsulation of the interface between the agents and native services on the host. CONFIDANT's agent gateway enables the agents to access the host filesystem directly to mitigate this exposure as described previously. The second attack considered is *Sugarcoating* of unfavorable reports. This 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. The third spoofing-based attack considered is *Recanting* of alert notification. CONFIDANT mitigates Recanting by enforcing transaction interlocks between agents. Agents must remain in constant communication. If agent communication is interrupted and handshaking is not maintained, then a suspicious activity has occurred which activates an alert.

4.2 Termination-based Tampering

Disabling an IDS sensor is called *Blindfolding*. This is mitigated in CONFIDANT by enabling multiple agents to perform similar tasks. Agent a_i^1 remains in communication with all agents $a_{j\neq i}^1$ in committee C^1. If Δt_r is exceeded then agent a_i^1 is determined to be missing by not maintaining an appropriate communication channel, or if an agent gateway cannot be contacted then an alert is initiated.

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. Multiple agents in each committee C^i perform the same functionality to mitigate tampering at any single point. Transactions between agents within a committee C^i, as well as between overlapping committees, are interlocked and are also 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 are expected and not received upon the expiration of the Δt_r window, an alert is initiated. Each gateway g_i provides agents with direct access to system resources so that alert notification is reliably transmitted to the security administrator.

4.3 Sidetracking-based Tampering

Some frameworks are subject to *Blockading*, or isolating a sensor from needed access to a component or data. 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 node is considered suspect and an alert is initiated. Architectures with centralized control are particularly vulnerable as Blockading can be focused on a single point either at the network interface, TP_N, or at the host process level, TP_{PT}.

Altering execution rates, or *Pacing*, is mitigated in CONFIDANT by redundancy of agents in committees C^i. Multiple committee agents traverse the network to perform analysis of local files on visited hosts. Messages are then passed across the network between agents on distributed hosts. Agent actions are based on internal timers defined by individual agents and not on the time of day, thus mitigating tampering at TP_{SC}. All agents a_j^i must provide status messages to cooperating agents in committee C^i prior to expiration of Δt_r, or else 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 committee C^i agents to pursue each simultaneous alert independently so that multiple alerts can be processed concurrently.

4.4 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 at TP_{ID}. This is mitigated in CONFIDANT by maintaining baseline data within each agent responsible for file integrity verification. When an agent a_1^1 computes a cryptographic digest for a file, the result is compared to internal baseline data encapsulated within multiple mobile agents and transmitted via events to agents in C^1. If the internal data is modified, agent redundancy enables file verification to be performed by other agents. An important consideration is the total size of baseline data maintained within each agent. If an agent is responsible for monitoring hundreds of files, the internal baseline size may eliminate any network efficiency benefits that mobile agents provide. One possible solution is to transmit a reference value for the individual baseline databases locally stored on the remote nodes. If the baseline database reference value is consistent with the internal data, individual system files are reliably be inspected for modification. If

Table 2: CONFIDANT approaches for tampering mitigation

Tampering Mode	Tampering Description	Mitigation Approach	Mitigation Description
Spoonfeeding	Alternate data stream is conveyed during file scan	Encapsulation	Vulnerable File I/O contained inside agents
Sugarcoating	Unfavorable cryptographic digest is modified to appear as the desired result	Validated transactions	SSL used for messaging and transport
Recanting	Fraudulent command is issued to deactivate alert	Interlocks, scrambling	Agent interactions are interlocked and spatially distributed
Blindfolding	Detection mechanism is disabled	Redundancy, vulnerability seeding	Exceptions are inserted to test detection status
Commandeering	Decision-making process is usurped	Interlocks, scrambling	Agent interactions are interlocked, and both spatially and temporally distributed
Soundproofing	Notification mechanism is eliminated or muted	Redundancy, interlocks	Alarms at each node with interlocked I/O
Blockading	Resource usage is forestalled to starve access	Pulse-taking	Interlocked file bandwidth monitoring and alert mechanism
Pacing	Scan timing reference is corrupted or execution priority is overwhelmingly reduced	Pulse-taking	Interlocked CPU throughput monitoring and alert mechanism
Scapegoating	Attention is diverted to a contrived distraction	Redundancy	Concurrent tracking via multiple agents
Retroactive Baselining	Reference values for digests are modified	Distinct Inception	Data dispatched with agents upon configuration
Descoping	Exemption is added to policy file to exclude scan coverage of unauthorized modifications	Mandatory Obsolescence	Configure only upon initial startup then destroy configuration agents
Value Jamming	Stand-alone process continuously writes `FALSE` into the memory location of status indicator	Redundancy, scrambling	Agent execution is spatially and temporally scrambled
File Juggling	Target files interchanged before and after scanning	Redundancy, scrambling	Unpredictable redundant scan scheduling

the baseline database reference value is incongruent with the internal data, an alert message is raised. The ability to monitor a large number of files while maintaining efficiency benefits of a mobile agent solution is the subject of future research.

IDS control components are subject to *Descoping* by tampering with the initial policy configuration data. As with Retroactive Baselining, this occurs at TP_{ID}. This is mitigated in CONFIDANT by including policy information within each agent. Agent a_1^1 contains the list of monitored files as defined prior to initial dispatch. The internal data is not modified during network traversal. If the policy information for a critical file is somehow maliciously altered within a_1^1, redundancy of agents in committee C^1 ensures that particular file will be inspected by other agents. Also, if policy data has been maliciously altered, the absence of messages provided by a_1^1 to other members of C^1 reflect that a monitored file was not scanned. This is detected by agents in C^1 as tampering.

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 committee agent to be responsible for maintaining file status information. Multiple agents reside simultaneously on a node at any given time, so there is no single memory location that serves as a vulnerable status flag. Memory locations used for status information can also vary each time an agent a_j^i visits a node due to occupying a different memory location. Since status flag memory locations can be both spatially and temporally distributed for each agent visitation, CONFIDANT is less vulnerable to tampering by jamming.

4.5 Selective Deception

In order for a framework to be subject to tampering at TP_{FS} by *File Juggling*, an adversary must be able to predict that a file integrity scan will occur at time t_{scan} as to perform undetected file system modifications. Selective Deception is mitigated in CONFIDANT by enabling multiple redundant agents a_j^i operating in committee C^i to have 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.

4.6 Tampering Mode User Rank

The defined user capability can be illustrated in terms of tampering modes applicable to each rank. As shown in

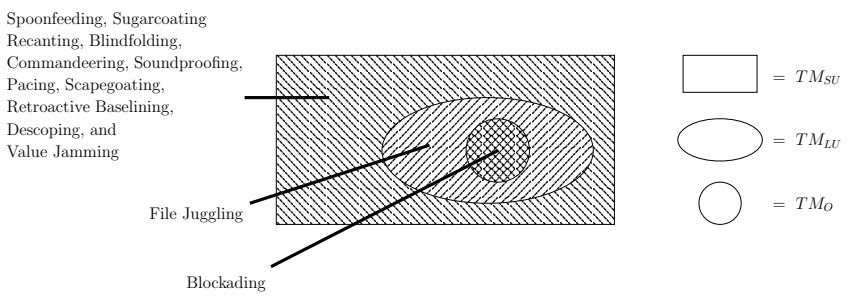

Figure 7: IDS tampering mode rank

Figure 7, outsiders exhibit exposures of the lowest rank, Rank III, and are able to perform only Blockading across the network. In addition to Rank III exposures, Legitimate Users can also perform File Juggling in certain circumstances. Consider the case of delegating web server administration responsibilities to someone who is not the superuser. The webmaster has the required access to modify web server configuration. If these configuration files are monitored as defined by the administrator in the IDS policy data, the webmaster has the ability to tamper via File Juggling. The remaining eleven exposures are restricted to Rank I, or the superuser level.

5 Conclusion

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. All capabilities are distributed and transactions are interlocked by tamper-evident handshaking protocols. Moreover, the agent *dispatch policies* and *travel itineraries* are constructed dynamically in response to events throughout the network.

It is necessary and sufficient to attain a robust file integrity framework by addressing tampering modes within the domain of administrators, denoted as TM_{SU}. By focusing development effort on superuser exposures, other vulnerabilities are mitigated without the need to explicitly design mechanisms to address exposures of lower rank. CONFIDANT employs mitigation strategies such as *Encapsulation*, *Interlocking*, *Redundancy*, and *Mandatory Obsolescence* for the defined tampering exposures.

Initial evaluation of CONFIDANT has shown it to be effective in mitigating several severe insider tampering exposures. Detailed evaluation methodology and results illustrating robust operation in the presence of single point-of-failure exposures are presented in [17]. Evaluation test cases for each insider tampering mode and a comparative metric scheme to evaluate the response of multiple frameworks to the same stimulus is presented in a companion paper. Future work includes investigation of scenarios that evaluate multiple simultaneous tampering modes and framework extensions to layered systems with secure hardware facilities.

References

[1] R. Anderson, T. Bozek, T. Logstaff, W. Meitzler, M. Skroch, and K. V. Wyk, "Research on mitigating the insider threat to information systems," in *Workshop Proceedings, RAND Corporation Report CF-163-DARPA*, pp. 5–15, ISBN: 0-8330-2962-2, Aug. 2000.

[2] 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.

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

[4] R. F. DeMara and A. J. Rocke, "Mitigation of network tampering using dynamic dispatch of mobile agents," *Computers & Security*, vol. 23, no. 1, pp. 31–42, 2004.

[5] S. Y. Foo and M. Arradondo, "Mobile agents for computer intrusion detection," in *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, pp. 517–521, 2004.

[6] Y. Jianga, Z. Xiab, and S. Zhanga, "A novel defense model for dynamic topology network based on mobile agent," *Microprocessors and Microsystems*, vol. 29, pp. 289–297, 2005.

[7] O. Kachirski and R. Guha, "Effective intrusion detection using multiple sensors in wireless ad hoc networks," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 57.1–57.2, 2003.

[8] S. Kent, "On the trail of intrusions into information systems," *IEEE Spectrum*, vol. 37, no. 12, pp. 52–56, 2000.

[9] G. H. Kim and E. H. Spafford, "Experiences with tripwire: the evaluation and writing of a security tool," in *Proceedings of the USENIX Applications Development Symposium*, pp. 3.1.1–3.1.7, Toronto, Ontario, Canada, 1994.

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

[11] C. Li, Q. Song, and C. Zhang, "MA-IDS architecture for distributed intrusion detection using mobile agents," in *Proceedings of the 2nd International Conference on Information Technology for Application*, pp. 451–455, 2004.

[12] T. Linden, *Nabou system integrity monitor*, http://www.daemon.de/Nabou.

[13] D. G. Marks, P. Mell, and M. Stinson, "Optimizing the scalability of network intrusion detection systems using mobile agents," *Journal of Network and Systems Management*, vol. 12, no. 1, pp. 95–110, 2004.

[14] J. McHugh, A. Christie, and J. Allen, "Defending yourself: the role of intrusion detection systems," *IEEE Software*, vol. 17, no. 5, pp. 42–51, 2000.

[15] J. McHugh, A. Christie, and J. Allen, *Intrusion detection: Implementation and operational issues*, Software Engineering Institute Computer Emergency Response Team White Paper, 2001), http://www.stsc.hill.af.mil/crosstalk/2001/01/mchugh.html.

[16] P. G. Neumann and P. A. Porras, "Experience with EMERALD to date," in *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA, USA, pp. 73–80, 1999.

[17] A. J. Rocke and R. F. DeMara, *CONFIDANT: Collaborative object notification framework for insider defense using autonomous network transactions*, *Autonomous Agents and Multi-Agent Systems*, in press.

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

[19] Rocksoft, *Veracity - nothing can change without you knowing: data integrity assurance*, http://www.rocksoft.com/veracity/.

[20] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for intrusion detection and response," in *DARPA Information Survivability Conference and Exposition*, 2000. DISCEX '00 2, pp. 3–11, 1999.

[21] Tripwire, Inc., *Tripwire for Servers for Security & Network Management*, http://www.tripwire.com/products/servers.

[22] *Xintegrity – Data Integrity Assurance and Network Security*, http://www.xintegrity.com/.

Adam J. Rocke received the Ph.D. degree in Computer Engineering from the University of Central Florida in 2004. He has been a lecturer and researcher at the University of Central Florida in the areas of system software and network security. Dr. Rocke's research interests are in distributed processing, network security, and embedded systems.

Ronald F. DeMara received the Ph.D. degree in Computer Engineering from the University of Southern California in 1992. Since 1993, he has been a full-time faculty at the University of Central Florida in the Department of Electrical and Computer Engineering. Dr. DeMara's research interests are in Distributed Processing and Special-Purpose Architectures. He is the editor of two books and has approximately 100 publications in-print or online on these topics. His research has been sponsored by the National Science Foundation, NASA, U.S. Army and Navy, National Security Agency, Harris Computer Systems, Lockheed Martin Information Systems, Theseus Logic Incorporated and others. He is a Senior Member of IEEE and a Member of ACM and ASEE. He has served on the Editorial Boards of the Journal of Circuits, Systems, and Computers, the journal Microprocessors and Microsystems, and IEEE Transactions on VLSI Systems.