Mobile Agent Data Integrity Using Multi-agent Architecture[*]

J. Todd McDonald[**]
Dept of Computer Science
Florida State University
Tallahassee, FL 32306-4530, USA
mcdonald@cs.fsu.edu

Alec Yasinsac
Dept of Computer Science
Florida State University
Tallahassee, FL 32306-4530, USA
yasinsac@cs.fsu.edu

Willard C. Thompson III
Dept of Computer Science
Florida State University
Tallahassee, FL 32306-4530, USA
wthompso@cs.fsu.edu

Abstract

Protection of agent data state and partial results in mobile agent systems continues to draw research interest. Current solutions to integrity attacks are geared at detection of malicious activity a posteriori. We propose multi-agent architecture that uses cooperating multi-hop and single-hop agents to prevent such attacks and discuss security features of our scheme. We also examine data protection services and the means for collecting agent data state in this context.

Key Words: mobile agents, security, data integrity, partial result protection

1 INTRODUCTION

Mobile agents are a method for implementing distributed systems and an emerging abstraction for architectural program design. Unresolved security issues continue to be cited among other reasons why mobile agents have not achieved widespread implementation [1]. Research continues to formulate mechanisms that secure the computational state of mobile agents in the presence of malicious hosts. Protecting an agent consists of two primary features: protecting the static executable code of an agent from disclosure or alteration and protecting the dynamic state of the agent as it incrementally changes during execution. We focus on the protection of the intermediate data results an agent gathers as it migrates through a network.

Agent data protection is concerned with keeping the state of an agent safe from observation (*confidentiality*) or keeping it safe from alteration (*integrity*) by malicious hosts. Integrity violations are typically only *detectable* after the agent returns to its origination, when it reaches an honest host in the itinerary, or when it stores partial results with a trusted third party. Our proposed architecture supports *prevention* of integrity violations (without data aggregation) and detection of these violations (with data aggregation) by using multiple cooperating agents to accomplish user tasks.

Essentially, we distinguish between the tasks of data *computation* and data *collection*, and assign these tasks to different classes of cooperating agents. This architecture provides security features ideal for protection against classical data integrity attacks such as insertion, deletion, and alteration of intermediate results. The novelty of our approach is in part the formalization of using different classes of agents to perform information computation and gathering. Essentially, our scheme relies on three cooperating types of agents: some agents performing computations, some managing tasks, and other agents carrying computational results back to the originating host.

The rest of the paper is outlined as follows. In section 2, we provide background and review existing literature. In section 3, we introduce separation of agent data *computation* from agent data *collection* and outline our approach to integrity of data using multi-agent architecture. Section 4 provides conclusions and a discussion of benefits and issues related to our approach.

2 RELATED WORK

Mobile agents are sent to gather information and perform tasks on behalf of a user, migrating to the source of information and reducing network bandwidth of messages required to process data. An agent is seen as a migrating program that has both a static part (code) and a dynamic state (data). Each host execution can be said to add new information progressively to the data state of an agent. Initial work in mobile agents such as [2] identified two modes of information gathering: stateless and stateful. In a stateless approach, agents can intermittently or at every hop send information acquired back home to the originator. In a stateful mode, the agent embodies in its data state the results of each host execution and carries with it a growing collection of information to each subsequent host in its itinerary. In this regard, we formalize in our architecture a mechanism by which both stateless and stateful integrity protection mechanisms can be incorporated into mobile agent interactions with the use of multiple classes of agents.

The use of multiple agents and transfer of partial results for safekeeping during agent migration are considered in other works as well. Roth in [3] proposed that an agent could transfer commitments to another cooperating agent that can verify and store the information gathered. In [3], agents are sent to disjoint sets of hosts and in turn can send each other commitments of current hosts via a host-provided secure communications channel. This provides a level of non-repudiation and requires a malicious host to corrupt other hosts that are on the co-operating agent's future itinerary.

[*]This material is based upon work supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number DAAD19-02-1-0235

[**]The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government

2.1 Partial Result Protection

Chained encapsulated results [4], partial result authentication codes [5], per-server digital signatures [5], append-only containers [6], and sliding encryption [7] are all examples of solutions from early mobile agent research that provide various levels of intermediate result protection. These frameworks use digital signatures, encryption, and hash functions in different combinations of chained relationships to provide detection and verification services. With these techniques, the originating host or an honest host in the path of an agent can identify when previous servers have inserted, truncated, or changed information from previous intermediate results carried by the agent. Loureiro et al. proposed a subsequent protocol in [8] that would allow a server to update its previous offer or bid. Roth has noted in [9] that many of these protocols suffer from replay and oracle attacks because they do not bind the dynamically collected data of an agent to its static code.

When malicious hosts collude, these protocols are weaker in detecting activity in the face of cooperating hosts that share secrets and information to change intermediate host results. Specifically, truncations are defined as integrity attacks where the data state of an agent is restored back to a state computed at a host that was previously visited in the itinerary. With the use of dynamically determined loose itineraries [3], detection of truncation attacks is difficult if not impossible in certain cases.

Vijil and Iyer in [10] augmented the append-only container of [6] with a means to detect mutual collusion and actually identify which hosts performed the tampering. Other recent work describes weaknesses of several protocols where truncations cannot be detected in certain cases. Maggi and Sisto in [11] provided a formal definition to describe protocol interactions in several different data protection mechanisms. In particular, they note that implementation of stronger forms of truncation resilience need to be implemented. Our notion for multi-agent architecture that separates data computation and data collection into different agent classes and services is inspired by this quest.

2.2 Multiple Agents/Fault Tolerance and Security

We propose to use multiple *classes* of agents that are given similar duties of either information computation or data gathering. The use of multiple agents is also part of the domain of fault tolerance, which seeks to provide guarantees on agent migration and task completion. Extensive work has already been done with integrating and providing fault tolerance in mobile agent systems [12, 13, 14]. Minsky et al. in [15] proposed that replicated agents and voting could be used to decide if malicious hosts have altered agent execution. Yee proposed a mechanism to detect replay attacks in [16] while Tan and Moreau extend an execution tracing framework in [17] to prevent denial of service attacks. In terms of multiple agents, Tate and Xu utilize multiple parallel agents that employ threshold cryptography to eliminate the need for a trusted third party in [18]. Tate and Xu also noted their work was one of the first to consider multi-agent settings on the basis of their security benefits. With our approach, we continue to analyze multiple agent architectures in mobile contexts on the basis of their security advantages.

A distinction exists between using the *same* agent logic replicated multiple times [15, 18] and using *different* agents to accomplish a single purpose-driven task [2], which our scheme utilizes. Work such as [19] has also tried to determine whether the use of multiple static agents is better than the use of multiple mobile agents from a performance perspective. Kotzanikolaou et al. in [20] presented architecture where a master agent and a set of multiple slave agents are used together to conduct electronic transactions. In [20], slave agents are mobile and travel to only one particular host to negotiate, but cannot complete a transaction without returning to the master agent. Our approach is similar in the sense that we conceptualize a master task agent that spawns and directs information gathering from multiple computation and collection agents, and then carries out any transaction logic separately.

2.3 Data Collection Services

Our architecture also uses *data bins* that allow an agent to leave computational results in an encrypted, retrievable form. Data lockers in [21] are described as a service provided for mobile users that keeps their data in secure and safe locations part of a fixed network. Our notion of a data bin is similar in that we envision a data service where intermediate results of agent computations can be safely stored during the transit of an agent through a network. Our approach is motivated by the security properties that a data bin offers instead of convenience or accessibility that is associated with a data locker.

3 MULTI-AGENT DATA INTEGRITY

The simplest method of preventing data integrity attacks is to not put partial or intermediate results in the hands of potential malicious servers. Reducing exposure or eliminating exposure of partial results all together is the essential idea behind our architecture. Specifically, we envision three different classes of agents: task agents, computation agents, and data collection agents. The *task* agent is responsible for the overall job a user wants to perform. *Computation* agents replicate in a fault-tolerant, single-hop manner or can be embodied in a single multi-hop agent to perform required computations. *Data collection* agents are responsible for providing ad-hoc or continuous data state collection. In essence, computation agents interact with hosts, under the direction of a master task agent, and leave partial results on the server where they are produced via the use of a data bin service. *Collection agents* are assigned the task of gathering and returning intermediate results to the master task agent that can then filter and make decisions based upon predefined criteria.

Our work involves prevention of manipulation, extraction, and truncation of information accumulated by an agent in a multi-hop free-roaming scenario. We do not address denial of service by a malicious host nor attempt to analyze whether a server has provided false information to the agent. We do not address privacy of execution (the knowability of a given function) or integrity of execution in terms of random code modifications. We assume that alterations to the static agent code are detectable by honest hosts when measures such as code signatures [5, 18] or execution tracing [22] are employed. We also assume that a public key infrastructure is in place or at a minimum the ability to distributed shared secrets among participants of the mobile agent system.

Various methods exist in the literature for formally describing the interaction of an agent with a host [8, 11, 18] and each sets forth data privacy characteristics that various protocols support. As illustrated in figure 1, an agent's execution can be described by the set of hosts that it visits, $\{h_1, h_2, \ldots, h_k\}$ and the associated set of data D, $\{d_1, d_2, \ldots, d_k\}$, that represents the incremental change in state of the agent as it visits each host and performs its task. This view of agent data interactions is part of the traditional idea of a competitive, electronic transaction application of mobile agents where bids or offers are collected by agents in various contexts, such as airline reservation [2, 4, 5, 23].

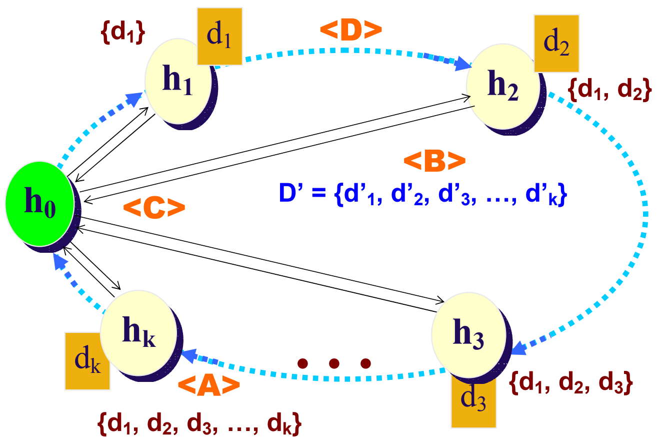

Figure 1: Single and Multi-Hop Agent Interactions

As shown in [11], this set of data can be described as being either a sequenced or unordered collection of the relevant computational results gathered by the agent as it traverses its itinerary and performs its task. Each host inserts a new result into the protected area of an agent in a manner that links the result to the previous results and/or the next host to be visited. Assuming that the originating host is h_0, the agent's path could also be described as the set $\{h_0, h_1, h_2, \ldots, h_k, h_0\}$. When an agent arrives back at its originating host, with task accomplished, the set of data results D', $\{d`_1, d_2, \ldots, d`_{k`}\}$, will represent the incremental changes in state of the agent as it migrates around the network. Figure 1 illustrates how the data set of a migrating agent essentially grows as each host executes the agent code, adding a data state to the protected area of a migrating agent.

The set D', indicated by figure 1, letter <B>, indicates what the agent *does* have on arrival back at its originating host and set D, indicated by figure 1, letter <A>, indicates the set of data results it *should* have. The highest level of protection that can be achieved is known as *strong data integrity* and is defined as the ability to detect whether set D, $\{d_1, d_2, \ldots, d_k\} \neq$ set D', $\{d`_1, d'_2, \ldots, d`_{k`}\}$ on return of the agent to the originating host. Strong data integrity would also include detection of all truncation attacks, which is still not possible in all cases when colluding hosts are involved.

3.1 Independent and Dependent Agent Data

Independent data can be viewed as a collection of offers, bids, or results that are used by an agent to perform some type of decision based on those gathered results. Independent data is seen in a single-hop agent (also referred to as remote code execution [22]) that migrates to a remote host, performs its operation, and then either sends its result back to the home platform or migrates back to the home platform carrying the result. In other words, the "result" of the agent is independent from the "result" gathered on any other host platform where the same computation is carried out.

When an agent migrates from host to host performing such a query or computation in a multi-hop mode, results of the current host are appended to previous results that are also carried by the agent. Figure 1, letter <C>, illustrates migration paths of an agent that uses single-hop logic, returning to the originating host after each execution. This type of computation can be performed by one single agent that makes *k* roundtrip migrations in a single-hop manner or by *k* agents that migrate to each host independently, where *k* is the number of agent servers. Data fusion is then performed after all hosts are visited and all query results are collected.

As an example of independent data, an agent that carries out a "sum" operation can do so by collecting inputs from a host and storing each value in some type of data collection. When the agent returns home, the values stored in the collection are added together to complete the operation. The multi-hop data state of the agent in this example depends on previous executions of the agent only in the sense that contents of the set of collected data must be carried forward faithfully from the previous host. In this case, truncations, insertions, and deletions are carried out by malicious hosts who modify the "values" that are carried by the agent. Data is considered *independent* because the code logic computes new data state by incrementally adding new results to the protected area of an agent. Data fusion or sorting is, again, conducted after all server results are collected.

In some agent applications, the computational result of the agent at state d_x is *dependent* on the computation result of the previous agent states $\{d_1 .. d_{x-1}\}$. Figure 1, letter <D> indicates the path of a multi-hop agent as it traverses a network, migrating from host to host carrying out computations. A multi-hop bidding agent can be designed to carry all of the bids for each host visited in its state and apply logic to determine the winner once all possible hosts are visited (thus utilizing *independent* data). The bidding agent can also be designed to carry the amount and identity of the lowest bidder in its state, which is updated along the way as the agent visits each host (*dependent* data).

In the context of our "sum" example, an agent with dependent data would only carry a sum variable that is updated by the input of each host in its route. The agent returns home with the sum calculated from the last host that it visited. Independent data is also referred to as data aggregation because a correlation exists between the previous and current execution state of the agent.

3.2 Multiple Agents with Independent Data

We present an architecture based on the interaction of three different classes of agents: *task agents*, *data computation agents*, and *data collection agents*. The *task* agent embodies the job an originating host wants to perform, such as a user's desire to purchase an airline ticket with a fixed set of criteria. *Task agents* spawn either a single multi-hop agent or multiple numbers of single-hop agents to perform computations in the form of information gathering or bid requests. *Task agents* spawn one or more *computation* agents with either fixed or free-roaming itineraries that visit host servers in a specific subject domain (such as airline reservation systems) and perform queries based on user criteria. *Computation agents* traverse the same route of a typical mobile agent and need to be uniquely identifiable to prevent the replay attacks expounded in [9]. As figure 2 illustrates, a task agent can remain at the originating host or be transferred to a trusted third party where computation agents are then spawned.

The agent framework in our architecture must be equipped with a data service, referred to as a *data bin*, which stores encapsulated data states of agents. These bins are similar to other notions of data lockers [21] or services that provide agent data safekeeping on trusted third party servers. Data bins have both a public part, where computation and collection agents can store and retrieve results, and a private part, where host-originating task agents can store partial results for later fusion. In independent data operations, computation agents do not arrive back at the originating hosts with a state payload containing a protected set of results. Instead, the computation agent leaves the result of its execution (embodied in the mutable state or as a query result) on each host via the public data bin service, protected with some form of agreed upon encryption scheme. Figure 2 depicts the spawning of the computation agent (a) that would visit hosts h1, h2, h3, and h4 with multi-hop migrations of {a1,a2,a3,a4,a5}.

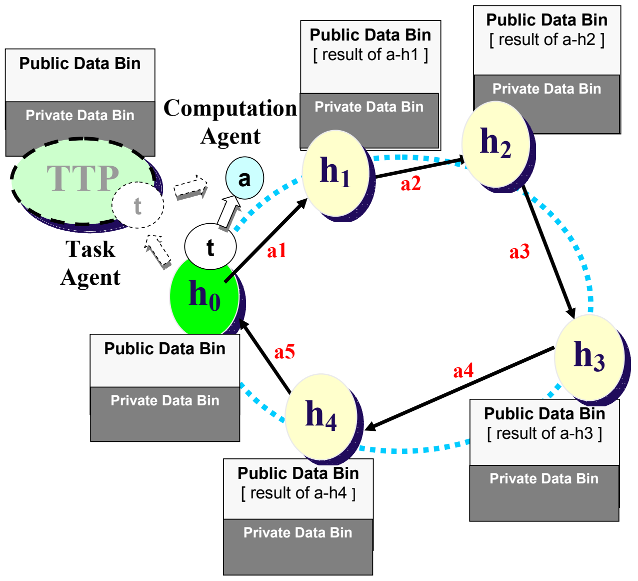

Figure 2: Activity of Task and Computation Agent

For greater fault tolerance, computation agents themselves can be launched in replicated mode as described in [15]. In either case, each computational result is linked to the identity of the agent and the identity of the originating host for later pickup. Data results can also be encapsulated using approaches such as [4,5,6,7]. At worst, a malicious host can only induce denial of service to the computation agent or only keep its own independent data result from being gathered by a collection agent. Authenticity and non-repudiability is achievable by binding the identification of the originating host and the unique agent identifier together with the agent data state. If the agent is single-hop, no data state is left and the agent returns to the originating host carrying the data state, as in remote evaluation operations [22].

Data collection agents are responsible for the single-hop mission of carrying back encapsulated data states or query results to the originating host. Figure 3 illustrates the activity of data collection agents that were spawned as a result of the activity of task agent (t) and its subsequent computation agent (a) seen in Figure 2. Each data collection agent (a,b,c,d in figure 3) stores its payload in the private data bin of the originating host and notifies the task agent of its arrival. In this aspect, data bins would provide private holding areas for data results that support information fusion of task agents that originate from that host as well as public holding areas where results of visiting agents are stored and are available for pickup.

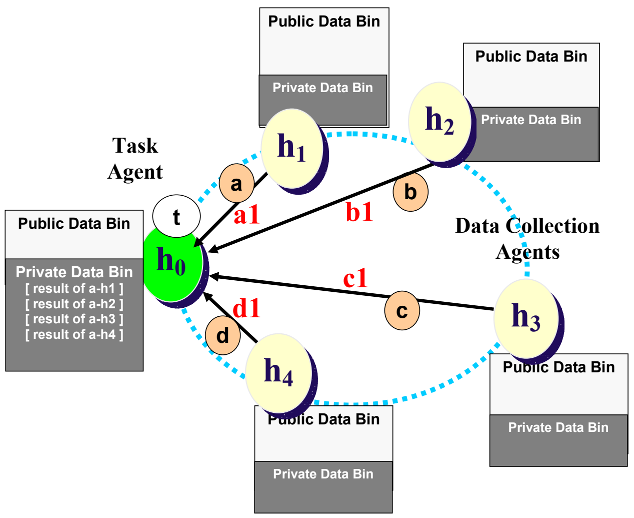

Figure 3: Activity of Data Collection Agents

Data collection agents are executed in one of three different configurations: 1) server-based response mode; 2) host-based request mode; and 3) autonomous data collection mode. In *server-based response mode*, each server visited by a computation agent spawns a data collection agent that performs an authenticated and encrypted single-hop transfer of the result (seen in figure 3). In the *host-based request mode*, the originating host sends data collection agents to each host that was part of the itinerary of the computation agent. The task agent does this in response to the completion of the computation agent. The *autonomous data collection mode* is a method where single-hop data collection agents are spawned either by a host that has just launched task agents or by agent servers who send results to trusted third party collection points on a recurring time interval. This approach is similar to a "garbage collection" service that runs in the background of a JAVA interpreter. With this method, data collection is built as a routine service that interfaces data bins with executing agent hosts.

Design of data collection protocols must ensure that only the originating host can retrieve its own data results, or in some altered form, that an agent can request previous data results for fusion at locations where a trusted computing environment is guarenteed. After all data collection activities have been accomplished and the task agent collates and filters results, it can spawn further computation agents that perform single-hop transactions or additional data gathering. In such cases where a single-hop transaction like a credit card billing is accomplished, no data collection is required. To summarize, all three classes of agents within our architecture are used in various combinations to accomplish a user task.

3.3 Multiple Agents with Dependent Data

Our proposed architecture relies on the general premise that agent *computations* should be separated from data state *collection* when a multi-hop agent is used. The static code of the computation agent has to *interact* with partial results of previous computations in order to produce a new data state result. To perform a multi-hop task that relies on *dependent* data, the data computation agent is modified to carry only the most recent state as its payload, while leaving a secured encrypted copy of its current state at each host server. Data collection agents in this configuration serve the role of a verification authority because the final data state of the agent can be compared against the incremental data states that are retrievable by collection activities. Detection of truncation violations is supported in this mode but not absolutely prevented when multiple colluding hosts are present.

Data aggregation implemented in this manner gives more freedom for using multi-hop agent logic. This architectural variation also resembles execution tracing proposed by Vigna in [22], but communications with the host in our scheme are meant to verify *integrity* of data (versus code execution) and are also automated (versus ad-hoc). The underlying data collection architecture itself can be used to incorporate or support other security measures such as execution tracing and data encapsulation [4,5,7].

3.4 Fault Tolerance and Security Issues

There are several fault tolerance issues that need to be addressed in our approach, just as in other schemes. For example, when storage space is exceeded in data bin services, some form of queue management is implemented (much like routers discard packets under certain load conditions). One or more trusted third parties can be used for data collection activities or task agent hosting (instead of the originating host) to allow for disconnected host operations. Timeout of task agents that must wait for results of both the computation agent and the data collection agents can be mitigated by providing time-based services that determine when agents have been unreasonably detained or diverted.

As with any multi-agent or mobile agent system, recovery from errors when messages are not delivered or when migration is not possible needs to be addressed. Failure of data bin services would require an alternative or default data storage service in the network if the host facility becomes unavailable. Failure of the original task agent, failure of one or more computation agents, and failure of data collection agents can be mitigated by such approaches as the shadow model of [14]. Other work on fault-tolerance such as [12,13,15] provide approaches to mitigate host failures and malicious activity.

Denial of service or random alterations of the code are not preventable because the agent server has ultimate power over an agent by having access to executable code and updatable state—though such activity can be detectable. When multi-hop agents with dependent (aggregated) data are used, the ability to mask or guard the function itself is needed to protect the computation agent against smart alterations of the code. We are currently researching other means to accomplish this aspect of agent protection [24] and plan to incorporate future results in consideration of multi-hop migrations. We also do not address the ability to keep keys used by both the computation and collection agent private, though it is an important issue with planned future research along the lines of work such as [25,26].

3.5 Performance

In most cases, adding security to a system always comes with a cost. Multiple interacting agents do bring more complexity and performance overhead to a system and the added benefits of security or fault tolerance has to be weighed against the increased communications costs. Performance issues boil down to the difference between a normal multi-hop agent that carries results with it and returns back to an originating host versus a static task agent that spawns one or more computation agents and receives responses from one or more collection agents.

A traditional migrating mobile agent that visits k servers would make $k+1$ migrations (see figure 1). The size of such an agent grows linearly according to the added data state and the size of the resulting query. When dependent data is designed into the agent logic, described earlier, the agent data state may not grow appreciably at all. For our architecture, there is now the overhead of single static task agent (present on the originating host or a trusted third party) and a computation agent that makes $k+1$ migrations (assuming a multi-hop traversal). At least k additional data collection agents must now make communication with data bin services and provide transport of results back to the host. The impact of doubling network transmission ($2k + 1$) and the increased consumption of resources due to the interaction of these three agent classes will be the subject of our continued future research.

4 CONCLUSION

Enforcing strong data integrity is the goal of many agent system schemes. We propose a multi-agent architecture for preventing data integrity attacks against mobile agents, especially truncations in the presence of multiple colluding hosts. Though the idea of communicating agent state to the originating host has been proposed previously, our architecture formalizes this approach in three classes of cooperating multiple agents and introduces the notion of a data bin service to facilitate data computation and collection activities.

We feel this approach offers several security benefits:

- It limits the impact any one agent platform can have on another
- No platform can influence previous computations during execution of the computation agent
- Impacting future computations in a multi-hop computation would require smart code alteration, which is the subject of other research in [24]
- Integrity attacks are reduced to denial of service

We rely on the assertion that results of an agent computation *must* be carried back at some point to an originating host. It is the exposure of these incremental results to possibly malicious hosts that motivates separation of *data computation* activities from *data collection* activities. Whether it is in the form of a modified agent state or as a combination of results that are embedded in a collection of agent state values, the agent either carries this set of data states within it or the state can be left at an agent platform and delivered by another, more secure, means.

The multi-agent approach also allows applications to be developed in a conceptual manner by leveraging the concept of agency while still providing a strong bound on data integrity and prevention of malicious host activity. The novelty of distinguishing between data *computation* and data *collection* to support data integrity in mobile agents will also provide continued avenues for future research and analysis.

5 REFERENCES

[1] G. Vigna, "Mobile Agents: Ten Reasons For Failure," Proceedings of MDM 2004, pp. 298-299 Berkeley, CA January 2004.

[2] D.M. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing," Journal IEEE Personal Communications, vol. 2, no. 5, pp. 34-49, October 1995.

[3] V. Roth, "Mutual protection of co--operating agents," in J. Vitek and C. Jensen, editors, Secure Internet Programming: Security Issues for Mobile and Distributed Objects, LNCS 1603, New York, NY, USA: Springer-Verlag, pp. 275-285, 1999.

[4] G. Karjoth, N. Asokan and C. Gulcu, "Protecting the computation results of freeroaming agents," in K. Rothermel and F. Hohl, editors, Proc. of the Second International Workshop, Mobile Agents 98, LNCS 1477, Springer-Verlag, pp. 195-207, 1998.

[5] B. Yee, "A sanctuary for mobile agents," in J. Vitek and C. Jensen, editors, Secure Internet Programming, volume 1603 in LNCS, pp. 261–274, New York, NY, USA, 1999. Springer-Verlag Inc.

[6] N. M. Karnik and A. R. Tripathi, "A security architecture for mobile agents in Ajanta," in Proceedings of 20th International Conference on Distributed Computing

Systems, pp. 402–409, IEEE Computer Society Press, 2000.

[7] A. Young and M. Yung, “Sliding Encryption: A Cryptographic Tool for mobile agents,” in Proceedings of the 4th International Workshop on Fast Software Encryption, FSE ’97. January 1997.

[8] S. Loureiro, R. Molva, and A. Pannetrat, “Secure Data Collection with Updates,” in Electronic Commerce Research Journal, 1/2:119-130, February/March 2001.

[9] V. Roth, “Empowering mobile software agents,” in Proc. 6th IEEE Mobile Agents Conference, LNCS Volume 2535, pages 47–63. Spinger Verlag, 2002.

[10] E.C. Vijil and S Iyer, “Identifying collusions: Co-operating malicious hosts in mobile agent itineraries,” in K.Fischer and D. Hutter, editors, Proc. of 2nd Intl. Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002), Bologna, Italy, July 2002.

[11] P. Maggi and R. Sisto, “A Configurable Mobile Agent Data Protection Protocol,” in Proc. of the 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), 2003.

[12] F. Schneider, “Towards fault tolerant and secure agentry,” in Proc. of the 11th Int. Worskhop on Distributed Algorithms, LNCS 1320, Springer-Verlag, Berlin Germany, 1997.

[13] H. Vogler, T. Hunklemann and M. Moschgath, “An Approach for Mobile Agent Security and Fault Tolerance Using Distributed Transactions,” in Proc. International Conference on Parallel and Distributed Systems (ICPADS’97), pp.268-274, Seoul, December 1997.

[14] S. Pears, J. Xu, and C. Boldyreff, “A Dynamic Shadow Approach for Mobile Agents to Survive Crash Failures,” Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03), pp. 113-120 ,Hakodate, Hokkaido, Japan, May 14 - 16, 2003.

[15] Y. Minsky, R. Renesse, F.B. Schneider, and S.D Stoller, “Cryptographic Support for Fault-Tolerant Distributed Computing,” in Proceedings of the Seventh ACM SIGOPS European Workshop, pp. 109-114, Connemara, Ireland, September 1996.

[16] B. Yee, “Monotonicity and Partial Results Protection for Mobile Agents,” in Proceedings of 23rd International Conference on Distributed Computing Systems, Providence, Rhode Island, May, 2003.

[17] H. Kim Tan, L. Moreau, “Extending execution tracing for mobile code security,” in Fischer, K. and Hutter, D., Eds. Proc. of 2nd Intl Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002), pages pp. 51-59, Bologna, Italy, July 2002. s Conference (NAECON), Dayton, OH, October 10-12, 2000.

[18] S.R. Tate and K. Xu, “Mobile Agent Security Through Multi-Agent Cryptographic Protocols,” in Proceedings of the 4th International Conference on Internet Computing (IC 2003), pp. 462-468, 2003.

[19] S. O'Malley, A. Self, and S. DeLoach, “Comparing Performance of Static versus Mobile Multiagent Systems,” in Proceedings of the National Aerospace and Electronic*s* Conference (NAECON) Dayton, OH, October 10-12, IEEE Press, pp. 282-289, 2000.

[20] P. Kotzanikolaou, G. Katsirelos, and V. Chrissikopoulos, “Mobile agents for secure electronic transactions,” in N. Mastorakis, editor, Recent Advances in Signal Processing Communications, World Scientific Engineering Society, pp. 363-368, 1999.

[21] Y. Villate, A. Illarramendi, and E. Pitoura, “Data Lockers: Mobile-Agent Based Middleware for the Security and Availability of Roaming Users Data,” in the 7th International Conference on Cooperative Information Systems (CoopIS 2000), Eilat, Israel, September 6-8, 2000, LNCS 1901, pp 275-286, Springer, 2000.

[22] G. Vigna, “Cryptographic traces for mobile agents,” in G. Vigna, editor, Mobile Agents and Security, LNCS 1419, Springer-Verlag, June 1998.

[23] J. Claessens, B. Preneel and J. Vandewalle, “(How) can mobile agents do secure electronic transactions on untrusted hosts? – A survey of the security issues and the current solutions,” ACM Transactions on Internet Technology. February 2003.

[24] W. Thompson, A. Yasinsac, J. McDonald. “Semantic Encryption Transformation Scheme,” to appear in Proc. of 2004 International Workshop on Security in Parallel and Distributed Systems, San Francisco, CA, September 15-17, 2004.

[25] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, “White-Box Cryptography and an AES Implementation”, Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002).

[26] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, “A White-Box DES Implementation for DRM Applications”, Proceedings of 2ndwork ACM Workshop on Digital Rights Management (DRM 2002), November 2002.