Extending execution tracing for mobile code security

Hock Kim Tan *
Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
hkvt99r@ecs.soton.ac.uk

Luc Moreau
Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
L.Moreau@ecs.soton.ac.uk

Keywords
Mobile agent security protocols, mobile agent security framework, cryptographic tracing

ABSTRACT
The problem of protecting mobile code from both denial-of-service and state tampering attacks by malicious hosts are not well addressed in existing techniques for mobile code security. We propose a possible approach based on extending an existing mobile code security technique: cryptographic tracing. This is achieved through the introduction of a trusted third party, the verification server, which undertakes the verification of execution traces on behalf of the agent owner. The interaction between the verification servers and host platforms in the new protocol is outlined. Security properties of the protocol are verified by modelling the system in CSP and checking the resulting state transitions using the model checker FDR. Limitations of this approach to verification are then briefly discussed.

1. INTRODUCTION
Mobile code security can be broadly classified into two areas: host security and code security. Host security is concerned with protecting the host platform (i.e. the computational environment that supports the execution of the agent) from malicious agents. Such agents may attempt to gain unauthorized access to local resources on the host or else inflict damage on other agents or programs executing on the host. Code security deals with the exact reverse; it attempts to safeguard honest agents from potentially malicious host platforms. Attacks from these malicious hosts could take the form of extracting confidential information (such as cryptographic keys or credit card numbers) embedded within the agent. Many viable mechanisms have been developed to tackle the host security aspect, but code security still remains problematic. An overview of security issues in both these areas, along with a comparative discussion of the current techniques available to address them can be found in [10], [4].

In general, the most common types of attacks on mobile agents described in literature can be classified as involving either:

- manipulation / extraction / truncation of information accumulated in the agent from its previous hops, particularly in a free-roaming scenario (i.e. where the itinerary of the agent is dynamically determined during migration). Techniques such as forward integrity [11] and chained signatures [5] can provide some protection again this type of attack by making it possible to detect the point in the route at which the attack occurred.

- alteration of the state or execution flow of the agent. Techniques such as execution tracing [24] or obfuscated code [9] are designed to either detect an attack and identify the perpetrator or render such attacks impractical by increasing the difficulty of interpreting the semantics of code execution correctly.

More recently the issue of resource control has become a topic of interest in host security research [25], [18], particularly in the Java environment which is extremely popular for developing mobile agent systems. This raises the interesting question from the viewpoint of code security: how do we ensure that an agent is provided sufficient resources by its host in order for it to complete execution successfully ? In its most basic form, a denial-of-service attack would involve a malicious host platform simply terminating all mobile agents that migrate to it. A more subtle form of attack could involve withholding resources (memory, CPU cycles) for a protracted period of time so that an agent executes for a longer period than it normally would. This may be problematic in certain situations; for example, if the agent owner is later charged for the amount of resources allegedly consumed by the agent on that host.

A survey of the code security literature reveals very few techniques that address this problem directly. The techniques we have mentioned so far appear to be vulnerable to this type of attack. Approaches that could address this problem include the use of replicated agents [17]or co-operating agents [19]. In [17], replicated agents are executed on different hosts and simple voting is used to determine the outcome of computational results. This approach is extended on in [19], where other strategies such as secret sharing, remote authorization or remote storage of commitments can be used as part of protocols involving two co-operating (but not necessarily identical) agents that communicate with each other while migrating in different host platform domains. Some

*This research is funded in part by QinetiQ and EPSRC Magnitude project (reference GR/N35816).

of the criticisms regarding these approaches are that they require replication of services on all host platforms and may fail if the number of malicious hosts outnumber the honest ones (for the case of replicated agents). Co-operating agents appear more feasible but require that a specific co-operating agent and associated protocol be created for each application scenario, thus making it difficult to use for generic mobile agents.

In this paper, we provide three contributions:

- Describe an approach to detecting some forms of denial-of-service attacks that involves extending the execution tracing protocol, an existing code security technique.
- Formally model the extended protocol using CSP and FDR and establish specific security properties.
- Outline some general problems related to the use of finite state models in modelling mobile code security protocols.

In the next section, we discuss the original protocol and how it is extended. The detailed protocol of message exchanges involved in this extension version is outlined in section 3. Formal modelling and verification of the protocol using CSP and the model checker FDR is presented in section 4. Section 5 discusses the limitations of this modelling approach as well as some general problems that may arise when attempting to formally model mobile agent security protocols. Finally, section 6 concludes with a short summary and direction for future work.

2. EXTENDING EXECUTION TRACING

In execution tracing (Fig. 1), a host platform executing an agent creates a trace of an agent's execution that contains precisely the lines of code that were executed by the mobile agent as well as all the external values that were read by the mobile agent. The trace is then stored by the host. This tracing activity is repeated for all hosts in the path of the agent. Upon its return, the agent owner may (if she/he suspects that the mobile agent was not correctly executed) request the complete trace of the agent's execution commencing from the first host platform (a). The agent owner will then simulate the execution of the mobile agent based on the information contained in the trace. This simulation will result in an intermediate state and identify the next host platform in the mobile agent's itinerary. The agent owner requests from this platform its trace (b) and proceeds in this manner for all hosts in the agent's itinerary (c). If at some point a discrepancy is found during the verification of the trace provided by a particular host platform, then a malicious host has been detected.

There have been some criticisms of this approach. The main drawbacks are the size and number of logs related to traces that need to be retained by the hosts, and the fact that the detection process is triggered only on suspicion that an agent has been manipulated. Other problems include the difficulty of tracing the execution results of multi-threaded agents.

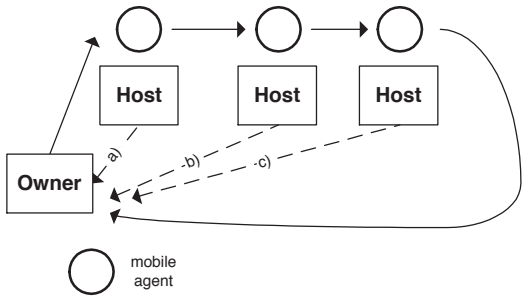

Figure 1: Execution tracing - original protocol

2.1 Introducing verification servers

In extending this protocol, we seek only to change the manner in which agents and traces are propagated in the system. The possible implementation of trace creation and verification using the approach outlined in the original protocol merits a complete analysis of its own and will not be discussed in this paper. Our approach is based on an earlier proposal [23] which involves the introduction of a trusted third party, the *verification server* that undertakes the process of verifying traces on behalf of the agent owner (Fig. 2). When an agent owner launches a mobile agent to a host platform (b), it creates a copy of the agent's code and state and forwards it onto a verification server (a) designated by the host platform. While the host executes the agent, it creates a trace of this execution simultaneously. Upon request of migration, the host then forwards this trace and the final agent state (c) to the designated verification server, which ensures that the execution sequence is valid. Once a verification server receives an agent copy, it will be aware of the identity of the platform executing the actual agent. It can thus implement a mechanism (for example, using time-outs) to ensure that a trace of the execution arrives from the required host within a reasonable time. This provides a way to safeguard against some forms of denial-of-service attacks.

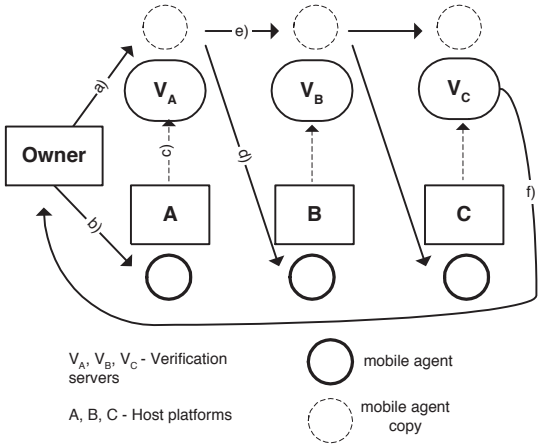

Figure 2: Execution tracing - extended protocol

When the validity of a trace is ascertained by a verification server, the agent is then forwarded from the verification server to its next destination host (d) and a copy (e) is sent to the corresponding verification server. Verification

and migration then proceed in this fashion until the agent completes its itinerary and returns to its owner (f). Host platforms do not need to retain traces once they are submitted to the verification server; verification servers only retain submitted traces which do not verify properly. These faulty traces can later be submitted as evidence to the agent owner or a suitable arbitrator for appropriate sanctions to be undertaken if so required. Only verification servers are allowed to migrate agents to host platforms; correspondingly, honest platforms will only accept agents from authenticated verification servers (more on this in section 4).

Prior to the commencement of a protocol run, host platforms will need to interact with verification servers to determine the servers that are willing (or capable) of verifying traces of agents executing in their environment. A host platform could thus have a choice of several verification servers to use in verifying any trace from its environment; conversely, a verification server could be responsible for verifying traces from several different hosts. Verification servers may delegate verification activities to other verification servers in the system if they are overloaded; this allows the formation of trust relationships between servers as detailed in [22].

2.2 Comparison with existing protocol

This extended protocol yields several advantages over the original one:

1. Trace verification is now performed by a verification server for each host platform that an agent migrates to in its itinerary. This permits the detection of malicious tampering as soon as it occurs at any platform on the agent's itinerary. In the original protocol, tracing only commences when an agent completes its tour and returns (by which time the damage inflicted by a malicious host could have been propagated to the remaining hosts in the itinerary) and even then, is only an optional activity triggered by a suspicious owner.

2. Traces have to be retained by a host in the original protocol (since the owner could request these for verification after a complete run of the agent), resulting in a high storage and maintenance overhead. This is no longer necessary in the extended version as verification is performed at every platform. In addition, verification servers can discard successfully verified traces and need only retain those with discrepancies as possible future evidence.

3. One of the primary motivations for using a mobile agent is to avoid communication problems attributable to low bandwidth or intermittent network links. In such an instance, the request of traces by an agent owner from potentially remote hosts in the original protocol could be problematic (for example, from behind a firewall). It would be easier instead for a host to select verification servers in its network vicinity that it can establish reliable communications with.

The extended protocol employs replication of agent code and state and is thus similar in motivation to the replicated agents approach. However, by imposing the trusted third party concept (i.e. the verification server is assumed to be a trustworthy entity that would not willingly collaborate with other hostile parties), we eliminate the need for replication of hosts as well as the possibility of failure that may arise when the number of malicious hosts outnumber honest ones in a voting scheme.

Our extended protocol shows greater similarity with the co-operating agents approach. The agent copy forwarded to verification servers for purposes of verifying the actual agent that migrates along a separate itinerary of host platforms can be regarded as a 'co-operating' agent that helps to detect tampering of the actual agent. However, the extended protocol offers the additional advantage of fault tolerance. A co-operating agent is not a replica of the actual agent to be protected, rather it is an agent that is specifically designed to support the actual agent in specific scenarios via constant communication so that it is immediately aware whenever the actual agent is compromised. In such an event however, it can only note or report the compromise but is incapable of continuing the compromised agent's agenda on its own. In the extended protocol however, if a verification server detects tampering in a trace, it can signal an exception to the agent copy executing in its environment so that suitable action can be taken.

3. PROTOCOL DESCRIPTION

In this section, we detail the protocol used in the extended version of execution tracing for one stage of a single protocol run (Fig. 3). A single protocol run is defined as the complete traversal of an unique agent instance along its itinerary, starting from where it departs from its owner (Fig. 2 b) to the point when it returns again (Fig. 2 e). This may include any possible loops in its path (i.e. when an agent returns to a previously visited host). We assume that a PKI is operating in the background, from which appropriate certificates and corresponding public keys can be obtained to perform encryption of data or verification of digital signatures. The pseudocode for the verification server and host platform is given in the Appendix. The format and sequence of messages exchanged in the protocol are shown in Fig. 4 and are explained as follows:

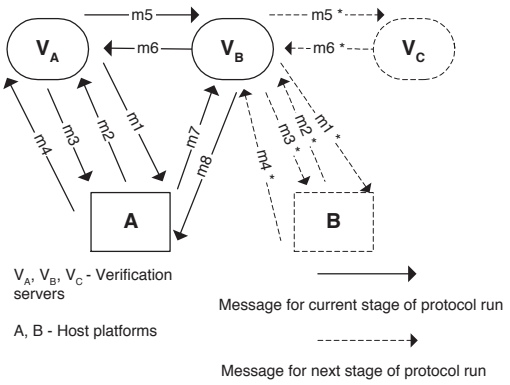

Figure 3: Message sequence in extended protocol

m_1: This message sent by V_A is in reaction to the mobile agent's request to migrate to A. It is essentially a request

m_1 : $\{\{V_A, A, n_{V_A}, Ic, \{c\}_{SPr(Owner)}\}_{SPr(V_A)}\}_{Pb(A)}$

m_2 : $\{\{A, V_A, t_{A_1}, n_{V_A}, Ic, Acc, V_B, \{A, V_B\}_{SPr(V_B)}\}_{SPr(A)}\}_{Pb(V_A)}$

m_3 : $\{\{V_A, A, t_{A_1}, n_{V_A}, Ic, S(c, V_A, T(c, V_A))\}_{SPr(V_A)}\}_{Pb(A)}$

m_4 : $\{\{A, V_A, t_{A_2}, n_{V_A}, Ic, S(c, V_A, T(c, V_A))\}_{SPr(A)}\}_{Pb(V_A)}$

m_5 : $\{\{V_A, V_B, \{c\}_{SPr(Owner)}\}_{SPr(V_A)}, m_4\}_{Pb(V_B)}$

m_6 : $\{V_B, V_A, n_{V_A}, Ic\}_{Pb(V_A)}$

m_7 : $\{\{A, V_B, Ic, n_{V_A}, T(c, A), S(c, A, T(c, A))\}_{SPr(A)}\}_{Pb(V_B)}$

m_8 : $\{V_B, A, n_{V_A}, Ic\}_{Pb(A)}$

Notation

- $\{X\}_{SPr(Y)}$ - indicates a message sequence X signed with the private key of entity Y
- $\{X\}_{Pb(Y)}$ - indicates a message sequence X encrypted with the public key of entity Y
- t_Y - indicates a timestamp created at entity Y
- n_Y - indicates a nonce created at entity Y
- Ic - refers to a unique agent identifier
- c - refers to static mobile agent code
- $T(c, Y)$ - refers to the trace of the agent code c after execution at entity Y
- $S(c, Y, T(c, X))$ - refers to the state of the agent code c after execution at entity Y, using the trace of the agent execution specified by $T(c, X)$

Figure 4: Protocol messages

to A to accept a mobile agent code instance Ic with associated code c. A nonce n_{V_A} is included here to keep track of a protocol run, and will be subsequently included in the following messages as a reference to that protocol run.

m_2: Upon receipt of m_1, A can decide whether or not to accept the mobile agent, based on consideration of the agent code. This may involve performing security checks on the code itself (i.e. host security) using techniques such as bytecode verification or proof carrying code. *Acc* in this message is therefore an indication of A's willingness (or otherwise) to accept the mobile agent. The last part of this message indicates the verification server (in this case V_B) that will verify the execution of any dispatched agent to A. This is accompanied by a certification signed by the verification server in question ($\{A, V_B\}_{SPr(V_B)}$). Message m_2 must be dispatched regardless of A's willingness to accept the mobile agent; this is necessary for V_A to distinguish between communication/server failure or agent rejection. A time-stamp t_{A_1} is included so that a record can be kept of m_2 in the event of a rejection.

m_3: An affirmative decision (with $Acc = Accept$) results in the state of the agent prior to migration being sent to A. In the event of rejection of platform A, the protocol run will terminate at this stage (with an appropriate exception flagged to the mobile agent), and recommence again at m_1 if so required by the mobile agent.

m_4: An acknowledgment message from A of the receipt of the agent. This message is vital to provide non-repudiation in the event that A attempts a denial of service attack once it has received the agent. The time-stamp t_{A_2} provides a reference value to implement a time-out mechanism in V_B to safeguard against denial of service.

m_5: A copy of the agent code and the entire contents of m_4 (which includes the agent's state) along with the appended signature created by $SPr(A)$ is then dispatched to V_B, the server that will be responsible for verifying correct execution on A. The identity of this verification server is obtained from the second portion of message m_2. Upon receipt of this message, a time-out mechanism will be in effect at V_B using t_{A_2} to ensure that m_6 arrives within a reasonable period of time, the failure of which is an indication of either a denial of service attack or a possible failure at A.

m_6: A simple acknowledgement of receipt of m_5 by V_B with inclusion of n_{V_A} and Ic to keep track of the current protocol run and agent instance.

m_7: Upon completion of agent execution, a trace is created at A ($T(c, A)$) and then submitted along with the new agent state $S(c, A, T(c, A))$ to the appropriate verification server.

m_8: A simple acknowledgement of receipt of m_7 by V_B with inclusion of n_{V_A} and Ic to keep track of the current protocol run and agent instance.

Upon receipt of m_7, V_B will commence replay of the agent c (identified by Ic) using the submitted trace $T(c, A)$. If the resulting state from this replay $S(c, V_B, T(c, A))$ is equivalent to the submitted state $S(c, A, T(c, A))$, then the next stage of the protocol run can be initiated, that is V_B can dispatch m_1* (the equivalent of m_1 in the next stage of the protocol) to B. The submitted trace can then be discarded. If equivalence is not obtained, an appropriate exception is raised to the mobile agent and the faulty trace and state is retained as evidence for further action by the home platform or an arbitrator (if so required).

4. FORMALLY MODELLING AND VERIFYING THE PROTOCOL

The primary security goal of execution tracing is to safeguard the state and execution flow of an agent, which is accomplished in the original protocol and in our extended version, by verifying agent states produced by replaying agents according to a given trace. If we assume that the verification and replay process is capable of detecting any malicious tampering, then the security goal essentially reduces to ensuring that traces and agents are dispatched correctly and securely to their designated destinations as outlined in the protocol. The original protocol uses various cryptographic primitives in order to achieve this goal (in a similar fashion to us) but does not attempt to formally verify the satisfaction of any security property. As it has been noted in literature that developing good security protocols is notoriously difficult [2], we believe that some form of modelling and verification is necessary in order to provide a basic assurance that certain specific security properties are achieveable. In the case of our extended protocol, we are primarily interested in two security properties that provide guarantee of correct and secure dispatch of agents and traces:

- Mutual authentication of verification servers and host platforms - It is important to make a distinction be-

tween these two entities as host platforms should only accept mobile agents that are dispatched from verification servers. This ensures that honest host platforms will never accept agents with potentially corrupted states directly from other platforms. The possibility of a hostile host platform spawning multiple copies of an agent and dispatching it randomly to other platforms in the system is also circumvented. In a similar context, a verification server has to ensure that it receives a copy of an agent from an authentic verification server to ensure that it is verifying the correct agent instance for a particular protocol run.

- Non-repudiation of commitment to executing agents - It is important to retain evidence of the fact that a host platform has committed to executing a particular mobile agent instance in a given protocol run to prevent a denial of service attack (i.e. terminating a mobile agent or delaying its execution for an inordinate period of time). This is primarily achieved by a digital signature appended to m_4 which is retained by V_A. The trace of agent execution as encapsulated in m_7 is also retained by V_B in the event it turns out to be faulty; this can be later be submitted to a third party arbitrator or the agent owner for sanctions to be undertaken towards the erring platform if the need arises.

In considering the security properties of the protocol, it should be mentioned that the basic underlying assumption is that the verification server is treated as a trusted third party. Thus we assume that verification servers will not engage in any action that will directly or indirectly lead to the corruption of an agent's state. We also make the usual assumption that the basic cryptographic primitives used are resistant to standard cryptanalysis and that private keys are not compromised. There are many approaches available for formally modelling and verifying properties of a security protocol. Some of the more commonly utilised ones include:

1. BAN logic of authentication [3], which reasons about the states and beliefs of agents involved in a protocol run and how these beliefs evolve with the reception of new information
2. Spi-calculus [1], which is an extension of π-calculus designed to deal with cryptographic primitives
3. Strand-spaces approach [6] uses the concept of a strand to represent the sequence of actions in which a particular protocol principal may participate and then reasons about how the strands interact or intertwine as participants interact by the exchange of messages
4. CSP-based approach [12] models the protocol interactions as a system described by CSP process algebra [8], for which violation of given specifications can be detected through the use of a finite-state model checker such as FDR [16]

We have chosen the last approach as it has been used successfully in discovering attacks upon cryptographic protocols ([12], [13], [15]). In addition, modelling protocol runs as interactions between entities using a process algebra like CSP appears to be a reasonably intuitive one. As a complement to this approach, tools such as Casper [14] have been developed that are capable of converting a high-level description of a security protocol to a CSP specification of the model that can be fed as input into the FDR model checker for subsequent verification. This greatly simplifies the process of CSP specification, which can be tedious and error-prone for complicated protocols. We employ Casper in describing our protocol in this section, further details on this protocol specification language can be found at [14].

4.1 Modelling the protocol in Casper

For the free variables section, we have declared the following variable types:

```
#Free variables

A : Agent
VA, VB : Server
nva, nt1, nt2 : Nonce
ca : AgentCode
asva, asa : AgentState
ata, atvb : AgentTrace
PK : Agent -> PublicKey
SK : Agent -> SecretKey
SSK : Server -> ServerSecretKey
SPK : Server -> ServerPublicKey
hash : HashFunction
InverseKeys = (PK,SK), (SSK, SPK)
```

The agent code c, trace $T(c, Y)$ and state of the agent $S(c, Y, T(c, X))$ (see Fig. 4) are represented by the types `Agent Code`, `AgentState` and `AgentTrace` respectively and can assume different values independently of each other. This makes the protocol easier to model as it is difficult to define multi-variable functions correctly using Casper (i.e. the state of an executed agent would be a one way function of its code, initial state and trace). A side effect of this is that the chances for attack by a malicious host is increased since it can interleave different values of code, state and trace with impunity in a protocol run. Indirectly, this enhances the strength of the security property that we intend to establish. We distinguish between the public and private keys of host platforms and verification servers as we regard them as two distinct classes of entities in our protocol. A hash function is used to model the unique agent identifier, Ic, which we treat simply as a hash of the agent code (the actual value of Ic is explained at the conclusion of Section 6).

There are three processes representing V_A (SERVERINITIATOR), V_B (SERVERRESPONDER) and A (HOSTRESPONDER) respectively of Fig. 3. All three entities will have knowledge of their respective secret keys and will be able to access the public keys of the other entities.

```
#Processes

SERVERRESPONDER(VB) knows SSK(VB), SPK, PK
SERVERINITIATOR(VA, nva, ca, asva) knows SSK(VA), SPK, PK
HOSTRESPONDER(A, nt1, nt2, asa, ata) knows SK(A), SPK, PK
```

The protocol is modelled below, where lines $1-8$ correspond to $m_1 - m_8$ of the protocol. We use nt1 and nt2 (of type

Nonce) to represent the time-stamps t_{A_1} and t_{A_2} issued by A, as we are only interested in their unique values in the protocol run and do not employ them to enforce a notion of freshness. The protocol run is preceded with step 0c., which establishes the result of an earlier interaction where A obtains a certification from a verification server V_B certifying V_B's capability of verifying agent traces from A. The % notation is used to indicate that this certification is not processed directly by A, rather stored in a temporary variable and then later relayed to V_A in message 2. The same comments apply as well to enc in message 4 and 5.

```
#Protocol description

0.    -> VA : A
0a.   -> A : VA, VB
0b.  -> VB : A
0c.  VB -> A : {{A, VB}{SSK(VB)} % storecert}{PK(A)}
1.  VA -> A : {{VA, A, nva, ca, hash(ca)}{SSK(VA)}}{PK(A)}
2.  A -> VA : {{A, VA, nva, nt1, hash(ca), VB,
               storecert % {A, VB}{SSK(VB)}}{SK(A)}}{SPK(VA)}
3.  VA -> A : {{VA, A, nt1, hash(ca), asva}{SSK(VA)}}{PK(A)}
4. A -> VA : {{A, VA, nt2, nva, hash(ca), asva}{SK(A)},
              {A, VA, nt2, nva, hash(ca), asva}
              {SK(A)} % enc}{SPK(VA)}
5. VA -> VB : {{VA, VB, ca}{SSK(VA)},
               enc % {A, VA, nt2, nva, hash(ca), asva}
               {SK(A)}}{SPK(VB)}
6. VB -> VA : {VB, VA, nva, nt2}{SPK(VA)}
7. A -> VB : {{A, VB, hash(ca), nva, ata, asa}{SK(A)}}{SPK(VB)}
8. VB -> A : {VB, A, nva, hash(ca)}{PK(A)}
```

We assume that the intruder is capable of creating its own agent trace, code and state. In addition, in line with the normal assumptions for an intruder in Casper, the intruder will also be capable of creating its own nonces and accessing the public keys and identities of all entities in the system.

```
#Intruder Information

Intruder = BadHost
IntruderKnowledge = {FirstServer, SecondServer, BadHost, Nb,
          Nbt1, Nbt2, Cb, PK, SPK, Asb, Atb, SK(BadHost)}
```

4.2 Specifying security properties

As mentioned earlier, the two important security properties to be established are mutual authentication and non-repudiation. We employ the concept of authentication as outlined in Casper . This is briefly expressed in the form of the statement Agreement(A, B, [x]), which states that A is authenticated to B on the basis of the fact that both A and B agree on the value of x. More formally [14], this means that if B (taking the role of responder) completes a protocol run, apparently with A, using the data value x, then the same entity A (taking the role of initiator) has previously been running the protocol, apparently with B, using the same value x. In addition, each such run of B corresponds to a unique run of A. x is typically some unique data item (such as nonce or time-stamp) known only to A or B. Mutual authentication will therefore require the additional statement Agreement(B, A, [x]) to be verified as well.

For the case of V_A and A, we can claim that these two entities are properly authenticated to each other after the exchange of m_1 - m_4, if only these two entities agree on the values $Ic, t_{A_2}, n_{V_A}, S(c, V_A, T(c, V_A))$. Ic is necessary to provide reference to the unique agent instance, n_{V_A} provides reference to the current protocol run, t_{A_2} and $S(c, V_A, T(c, V_A))$ provides reference to A's response in m_4.

```
Agreement(VA, A, [nva, nt2, hash(ca), asva])
Agreement(A, VA, [nva, nt2, hash(ca), asva])
```

Similarly mutual authentication between V_A and V_B and between V_B and A can be expressed as

```
Agreement(VA,VB, [nva, nt2])
Agreement(VB,VA, [nva, nt2])
Agreement(VB, A, [nva, nt2])
Agreement(A, VB, [nva, nt2])
```

Non-repudiation can be simplified to the more general case of maintaining secrecy of specific data items whose non-repudiation is to be established. If we know that only entity A issues item x in a protocol exchange between itself and another entity B, and if we can establish that item x remains secret in such a protocol exchange, then we can conclude that A is indeed responsible for issuing x. In our protocol, we are not interested whether x can later be duplicated by another entity (such as B) in another protocol run, rather we are concerned with whether x is issued in a given protocol run. The property of non-repudiation then follows simply by applying a digital signature to x. As mentioned earlier, non-repudiation is necessary for messages:

1. m_2 - to provide evidence a host accepts or denies an agent for a particular protocol run indicated by n_{V_A} and an agent instance indicated by Ic;
2. m_4 - to provide evidence a host has received the state $S(c, V_A, T(c, V_A))$ necessary to begin execution of the agent instance Ic in the protocol run indicated by n_{V_A};
3. m_7 - to provide evidence the agent instance was Ic executed with a trace $T(c, A)$ to provide a state $S(c, A, T(c, A))$.

In Casper, the statement Secret(A, x, [B]) is used to express the property that A believes x remains secret in an interaction between itself and an entity that appears to be B. If this entity is not B, then x will remain hidden to it. Thus, to show non-repudiation, we have the following security specifications [1]:

```
Secret(VA, nva, asva, ca, [A])
Secret(A, nt1, [VA])
Secret(A, ata, asa,  [VB])
Secret(VB, nt2, [A])
```

Both specifications for mutual authentication and secrecy were satisfied in the resulting CSP model that was checked using FDR. The checking process itself was lengthy (several hours) due to the complexity of the protocol and the number of independent data variables involved.

[1] These are provided in an abbreviated form; specifications for secrecy should be in the form Secret(VA, x, [A]) for each item x

5. LIMITATIONS OF MODELLING USING CASPER AND FDR

The use of Casper to model security protocols for mobile agent systems has been attempted previously by Hannotin et al. [7]. In their work, they attempt to verify the property of data integrity in a protocol proposed by Corradi et. al [5] which intends to safeguard data accumulated by a mobile agent (for example, price offers from various shop platforms) during its itinerary from invalid tampering. The protocol functions by making tampering of this data by a malicious host (for example:- modification or truncation of previous offers) detectable by either the home platform of the agent or the next honest host that the agent migrates to. Although this property is verified in the CSP model that they develop, the same protocol (as well as other protocols with a similar motivation of protecting accumulated data) was shown to be vulnerable to a certain type of attack described by Roth in [20], which has a general two-step approach:

1. Protocol data from an honest mobile agent is cut and pasted on to a 'dummy' mobile agent generated by a hostile host. This mobile agent is then launched to another honest host with which it interacts in a certain manner to acquire critical information about the current protocol run (which it would not normally be able to acquire without the use of the 'stolen' protocol data).

2. The 'dummy' agent migrates back to the hostile host with this information, which is then used by the host in some way to change the accumulated data of the honest agent. This change will subsequently be undetectable when the honest agent is migrated on to the next host or back to its home.

The strategy of this attack is not new and is similiar in motivation to an earlier well-known attack on the Needham-Schroeder public key protocol described by Lowe [13]. In his attack, a replayed message from a previous protocol run is used by an intruder to initiate a new protocol run in which an unsuspecting participant is then abused as an unwitting oracle to reveal confidential information. This information can then be used to compromise the integrity of a communication channel. Roth's technique is essentially the same with the primary difference being that a new mobile agent (instead of a replayed message) is used by a malicious host in a new protocol run to initiate the oracle attack. In order to nullify the attack, it is necessary to prevent one or both of these steps from occurring. Roth presents a method to prevent the first step by using authentication to uniquely associate the identity of an agent instance along with the protocol data transported by it. This would allow an honest host to discern whether an incoming agent is carrying protocol data that belongs to it or that was 'stolen' from another agent. The host could then refuse to accept or execute agents carrying 'stolen' data, thus preventing itself from being abused as an oracle. This security measure is in actual fact a form of host security, and we have here an interesting illustration of how the two different aspects of code and host security (which often appear to be orthogonal to each other) can be actually closely interlinked.

The main reason underlying the failure of Hannotin's Casper model to detect such attacks is the inability to model a mobile agent accurately using traditional cryptographic protocol analysis methods. In those methods, a fundamental assumption in analysis is that the format of messages exchanged between static entities and the sequence in which they occur within a single protocol run are predefined and remain fixed throughout the duration of the protocol run. Thus, attacks can only occur through judicious interleaving, reflection or replay of messages from different protocol runs. In Hannotin's approach (and our approach as well) the mobile agent is implicitly treated as a unique, static portion of a message. This permits a reasonably straightforward approach to modelling, but as we have just seen, it is not accurate as it does not reflect the ability of the mobile agent to potentially alter the sequence and content of messages during an ongoing protocol run. For example, in the second step of the attack, the data carried back by the 'dummy' agent to the hostile host has to be part of the specification of the protocol run (since this is the data that actually allows the hostile host to successfully carry out its attack). Obviously, the format and contents of this data cannot be predefined and will depend on the interaction of the agent with the honest host. Thus, in order for a model to be able to detect such attacks, two additional requirements are necessary:

1. The model must be able to encapsulate all possible behaviours of a mobile agent (as a function of its code, internal state and state of its execution environment) that have the ability to alter the format or sequence of messages exchanged within a single protocol run

2. The model must be able to take into account all these different possibilities of message contents and sequences when it is used to simulate a protocol run

With regards to the first requirement, the identification of the specific state or code of an agent that is capable of altering the format or sequence of messages is clearly not a trivial matter. Even if this could be accomplished, the additional possibilities for protocol runs with different message contents and sequences will greatly increase the number of possible interleaving of protocol runs, consequently creating a potential explosion in the state space to be explored. This may make it less suitable for use on a finite state space model checker such as FDR. In that case, checking the viability of the model may require techniques to reduce the state space explosion (such as those used in [21]) or modelling the protocol using a different approach (for example strand spaces or spi-calculus). As a matter of interest we note that the attacks described by Roth were discovered in an ad hoc, intuitive manner without resort to any formal methods of verification. It is thus possible that more subtle attacks may yet exist on the protocols in question (even after the remedy of authentication is applied), if these protocols can be expressed and analysed in more thorough manner using models that encapsulate the two requirements that we have briefly discussed.

Since we also treat the mobile agent as a static message in our approach, our model is equally susceptible to the same

vulnerabilities as Hannotin's. However, our modified protocol for execution tracing differs from the approach employed by Corradi as well the original execution tracing protocol in an important way: agent state (and code) is replicated. In our approach, the agent that is actually migrated on to the next host platform is the mobile agent copy on the verification server (the trusted platform), and not the actual agent on the current host. The consequence of this is that the current host will not be able to directly manipulate the state of the mobile agent, in effect nullifying step two of Roth's attack. The only way a hostile host can affect the state of the agent copy is through the trace it supplies; this however will also contain the signature of the host to act as a measure of non-repudiation. Supplying faulty traces as a form of attack is thus meaningless as liability can eventually be established for the resulting problems that arise (this, of course, is based on the assumption that the economic cost of being sanctioned for an attack is greater than the economic cost resulting from the attack). Therefore, our only concern is ensuring that traces, agent code and agent state are securely propagated in our system, and that traces are correctly associated with the corresponding agents. Mobile agent behaviour is subsequently of no concern to us any longer. We are therefore justified in using Casper to model our protocol as the nature of our protocol is now analogous to those modelled in standard cryptographic protocols. It is also our belief that completely secure code execution on untrusted platforms cannot be achieved without some form of code/state replication.

6. CONCLUSION

In this paper, we identified the need for code security techniques that address the concept of denial-of-service attacks in addition to the usual data integrity and state tampering attacks. A technique to detect some forms of such attacks is proposed which involves the extension of a well known code security technique, execution tracing. This essentially involves the introduction of a trusted third party, the verification server, that undertakes verification of traces on behalf of the agent owner. The advantages of this modified technique as compared to the original approach as well as other techniques that prevent denial-of-service attacks are outlined. The sequence of messages for the new protocol is described in detail, and is then modelled in CSP using the high-level security protocol description language, Casper. The model is then analysed in FDR to determine whether specific security specifications are valid. Finally, we discuss the limitations of modelling the protocol using Casper and finite state model checkers such as FDR and point out the difficulties involved in formal modelling of mobile agent security protocols in general.

Our current work focuses on developing a practical method to implement creation and verification of traces in a working mobile agent system. In addition, we are also looking at ways of reducing the cryptographic cost of the protocol without compromising on its security properties. A more formal method of expressing the use of time-outs to provide protection against denial-of-service attacks would also be useful. Once the protocol is sufficiently refined and trace verification properly developed, a mobile agent framework using the extended protocol can be created and evaluation conducted against existing code security techniques.

7. REFERENCES

[1] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] Martin Abadi and Roger M. Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering*, 22(1), 1996.

[3] Micheal Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871), 1989.

[4] D. M. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[5] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents integrity in e-commerce applications. In *Proceedings of the19th IEEE International Conference on Distributed Computing Systems Workshop, IEEE Computer Society Press, Austin*, May 1999.

[6] F. J. Thayer F'abrega, J. C. Herzog, and J. D. Guttman. Strand spaces : Why is a security protocol correct ? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, February 1998.

[7] Xavier Hannotin, Paolo Maggi, and Riccardo Sisto. Formal specification and verification of mobile agent data integrity properties : A case study. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

[8] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.

[9] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[10] Wayne Jansen. Countermeasures for mobile agent security. In *Computer Communications, Special Issue on Advances in Research and Application of Network Security*, November 2000.

[11] G. Karjoth and N. Asokan. Protecting the computation results of free-roaming agents. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[12] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, number 1055 in LNCS. Springer-Verlag, 1996.

[13] Gavin Lowe. Some new attacks on security protocols. In *9th IEEE Computer Security Foundations Workshop*, 1996.

[14] Gavin Lowe. Casper : A compiler for the analysis of security protocols. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[15] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[16] F.S.E. Ltd. Failures-Divergence Refinement : FDR2 User Manual, Available at http://www.formal.demon.co.uk/fdr2manual/. Technical report, Formal Systems Europe, 1999.

[17] Yaron Minsky, Robbert van Renesse, Fred B. Schneider, and Scott D. Stoller. Cryptographic support for fault-tolerant distributed computing. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, 1996.

[18] Luc Moreau and Christian Queinnec. Distributed computations driven by resource consumption. In *Proceedings of IEEE International Conference on Computer Languages (ICCL'98)*, 1998.

[19] Volker Roth. Mutual protection of co-operating agents. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer-Verlag, 1999.

[20] Volker Roth. On the robustness of some cryptographic protocols for mobile agent protection. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

[21] D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1), 2001.

[22] H. K. Tan and L. Moreau. Trust relationships in a mobile agent system. In *Proceedings of the 5th IEEE International Conference on Mobile Agents, Georgia, USA*, December 2001.

[23] H. K. Tan and L. Moreau. Certificates for mobile code security. In *Proceedings of the 17th ACM Symposium on Applied Computing*, March 2002.

[24] Giovanni Vigna. Cryptographic traces for mobile agents. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[25] Alex Villazon and Walter Binder. Portable resource reification in java-based mobile agent systems. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

APPENDIX

Operation of host platform

Accept m_1
Verify signature $SPr(V_A)$ and identity of verification server (V_A) through a certificate
Verify signature on agent code ($\{c\}_{SPr(Owner)}$) to detect possible tampering
Perform security check on agent code ($\{c\}$)
If decision is made to commit to running agent code
  Select verification server V_B to be employed in verifying the code
  Submit in m_2 suitable certification $\{A, V_B\}_{SPr(V_B)}$
  If verification server responds with m_3
    Verify signature on $S(m, V_A, T(m, V_A))$ to safeguard against possible tampering
    Respond with acknowledgment m_4
    Instanstiate agent code with verified state
    Commence execution of mobile agent and create a trace of its execution sequence $T(m, A)$
    Upon completion of an agent run, sign trace and submit in m_7 to V_B
  else
    Terminate protocol run and await commencement from another verification server
else
  Indicate refusal in reply m_2
  Terminate protocol run and await commencement from another verification server

Operation of verification server V_B

Receive initial state of agent $S(c, V_A, T(c, V_A))$ in m_5
Implement time-out mechanism using t_{A_2} from m_4
If trace $T(c, A)$ in m_7 arrives in specified time period
  Verify identity of host submitting trace
  Replay agent execution from initial state and submitted trace to obtain final state $S(c, V_B, T(c, A))$
  If final state $S(c, V_B, T(c, A))$ is equivalent to submitted state $S(c, A, T(c, A))$
    Identify destination platform contained in $S(c, V_B, T(c, A))$
    Submit m_1* to destination
    Receive m_2*
    If Acc in m_2* is positive
      Verify identity of the verification server associated with submitted certification
      Submit final state of agent $S(c, V_B, T(c, A))$ to host platform in m_3*
      Receive acknowledgment of reception m_4* from the host platform
      Forward $S(c, V_B, T(c, A))$ and $\{c\}_{SPr_{Owner}}$ on to the next verification server in m_5*)
    else
      Signal exception to agent
      Record platform identity that refused to host the agent
  else
    Retain trace $T(c, A)$ as evidence
    Signal exception to agent
    Record occurrence of trace inequivalence in agent
else
  Signal exception to agent
  Record occurrence of time-out in agent