Protecting Free-Roaming Mobile Agent against Multiple Colluded Truncation Attacks

S.Venkatesan
Department of CSE,
Anna University
Chennai-600025, India
venkalt_s@yahoo.co.in

C.Chellappan
Professor, Department of CSE
Anna University,
Chennai-600025, India
drcc@annauniv.edu

ABSTRACT
Mobile agent environment is one of the emerging technology to reduce the network traffic. Various security issues are identified and protected. Now the new security issue in this environment is the multiple colluded truncation attacks in the Free-roaming mobile agents. This paper proposes the identity verification mechanism to protect the multiple colluded truncation attacks. The identification is verified from the beginning dummy offer of the creator of the agent.

Categories and Subject Descriptors
C.2.0 [General]: Security and protection I.2.11 [Distributed Artificial Intelligence]: Intelligent Agent D.2.0 [General]: Protection Mechanism

General Terms
Security, Algorithms, Theory

Keywords
Free-Roaming mobile agent, colluded attacks, Truncation attacks, encapsulated offer

1.INTRODUCTION
Mobile agents are software programs to perform computation in various hosts and then bring the results to the owner of the agent. Roaming agents are moving from one node to another node by the respective statements of the owner but the free-roaming agents are moving from one host to another host without any respective migration paths by the owner, i.e., depends upon the requirements and the current conditions, the current host will select the next host for agent. In the free-roaming mobile agent environment, security is the main issue. That the malicious host may modify, delete or insert (malicious) data in the results of the mobile agents, which they collected from the previous hosts. For this types of attacks the various methods of prevention protocols are proposed and solved.

Nevertheless another one type of attack is raised is the multiple colluded attack, i.e. more than one host colluded together to discard the single data or the stream of data between the hosts. The general cryptographic mechanism is used for the protocols.

2.PREVIOUS WORKS
Yee[1] proposed the Partial Result Authentication Code(PRAC) to protect mobile agents results. Here the agent and its originator maintain a list of secret keys or a key generating function. The agent uses a key to encapsulate the collected offer and then destroys the key. However, a malicious host may keep the key or the key generating function. When the agent revisits the host or visits another host conspiring with it, a previous offer or series of offers would be modified, without being detected by the originator and also the carrying of list of keys by the free roaming agents for all the system is not possible.

Karjoth et al.[2] extends the above schemes for the efficient security purposes. Each host generates a signing key for its successor and certifies the corresponding verification key. Using the received signature/verification key pair, a host signs its partial result and certifies a new verification key of the next host. This technique will avoid the modification attack in above scheme, but not a two-colluder attack. In this attack two visited hosts can collude to discard the partial results collected between their respective visits. Here it uses digital signatures and hash functions to protect a chain relation.

Karnik et al.[3] This protocol uses an encrypted checksum to build a backward chain relation to link an agents previous result with the agents data generated at the currently visited host. It guarantees that only new data can be added to the results of the agent collected and no data can be deleted from them. It does not support two-colluder attacks. It is the compact method of the above scheme.

Corradi et al.[4] Here the protocol uses the backward and forward chaining. At each host, the protocol runs a hash function to compute a cryptographic proof of a result from the previous host, a result generated at the current host, and the identity of next hop. Like other protocols, this protocol cannot defend two-colluder truncation attacks.

Cheng et al.[5] This protocol provide the co-signing mechanism to defend the two-colluder truncation attacks. Here a preceding
host co-signs a result generated at the current host. Attackers need their preceding nonAttackers to co-sign fake offers when they launch two-colluder truncation attacks, and then their actions can be detected. Here also the publicly verifiable forward integrity propriety, this protocol generates a pair of one time secret private and public keys at each host for its successor.

Yao et al.[6] Songsiri[7] Here the Trusted Third Party is used to store the collected offers. To defend a stemming attack, a special loop. It requires its preceding host to co-sign encrypted data. The co-signers cannot check the encrypted data, malicious hosts may ask their preceding hosts to sign encrypted documents and use these signatures against the co-signers later. This protocol cannot defend multiple-colluder truncation attacks.

Darren et al.[11] This protocol will defend the two-colluder truncation attack with the help of the one hop backward and two-hop forward chaining method. And also it will defend the multiple colluder truncation attacks. It will avoid the fake stem attack and then the interleaving attack. It has the drawback that the two-colluders in the adjacent means it will not defend the attack.

Our new protocol addresses all the issues found in the previously available protocols, especially solutions to the multiple colluder truncation attacks. This paper is discussed as follows. In segment 3, we describe the common mobile agent security properties discussed in the previous papers. In segment 4, we analyze the new protocol and its mobile agent properties. In segment 5 &6, we analyze the protection for various attacks and at last in the segment 7, we conclude the specialty of this protocol.

3.NOTATIONS AND SECURITY PROPERTIES

We use the similar notations used in the other schemes [2][5][9][11].

<table>
<thead>
<tr>
<th>Table 1.Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notations</strong></td>
</tr>
<tr>
<td>S₀, S₁,…, Sₙ</td>
</tr>
<tr>
<td>O₀</td>
</tr>
<tr>
<td>S₀, S₁,…, Sₙ</td>
</tr>
<tr>
<td>r₀, r₁,…, rₙ</td>
</tr>
<tr>
<td>P₀, P₁,…, Pₙ</td>
</tr>
<tr>
<td>tP₀, tP₁,…, tPₙ</td>
</tr>
<tr>
<td>Encᵢ(m), H(m)</td>
</tr>
<tr>
<td>Sigᵢ(m), H(m)</td>
</tr>
</tbody>
</table>

Assume that the agent has a chain of encapsulated offers O₀, O₁,…, Oₙ. The following mobile agent security properties are based on assumption [2][5][11].

| **Data confidentiality** | Each offer Oᵢ is encrypted by Sᵢ’s public key Pbᵢ. Only the originator can decrypt the offer O₀. |
| **Non-repudiability** | Each offer Oᵢ is signed as Sigᵢ(m). Sᵢ cannot deny its offer Oᵢ after Sᵢ receives the offer and verifies the signature. |
| **Forward privacy** | None of the identities of the creators of offer Oᵢ can be extracted by anyone except the originator Sᵢ. |
| **Strong forward integrity** | Assume an attacker Sₙ holds encapsulation offers O₀, O₁,…, Oₙ₋₁, Oₙ, and modifies or replaces Oₙ with a new offer. Since Oₙ is intact, the chain relation hₙ=H(Oₙ₋₁, rₙ, Sₙ₋₁, Sₙ₋₂) in the encapsulated offer Oₙ. Since Oₙ is intact, the chain relation hₙ=H(Oₙ₋₁, rₙ, Sₙ₋₁, Sₙ₋₂) must be hold true, i.e. H(Oₙ₋₁, rₙ, Sₙ₋₁, Sₙ₋₂)= H(Oₙ₋₁, rₙ, Sₙ₋₁, Sₙ₋₂). This violates the assumption that the hash function H is collision-free. It is impossible for an attacker to modify or replace any offer without changing the next encapsulated offer if a collision free hash function is used in the protocol. |
| **Publicly verifiable forward integrity** | Any one can verify the encapsulated offers Oᵢ by checking whether the chain is valid at Oᵢ. |
| **Insertion defense** | No offer can be inserted at i unless explicitly allowed. |
| **Truncation defense** | Assume an attacker Sₙ truncates all encapsulated offers after Oₙ₋₁, and then appends its own offer Oₙ. The new chain of encapsulated offers is now O₀, O₁,…, Oₙ₋₁, Oₙ. Since Oᵢ is intact, the chain relation hᵢ=H(Oᵢ₋₁, rᵢ, Sᵢ₋₁, Sᵢ₋₂) must be hold true, i.e. H(Oᵢ₋₁, rᵢ, Sᵢ₋₁, Sᵢ₋₂)= H(Oᵢ₋₁, rᵢ, Sᵢ₋₁, Sᵢ₋₂). This violates the assumption that the hash function H is collision-free. |

4. PROPOSED PROTOCOL

Our protocol builds a chain relation to all the previous encapsulated offers and forwards it to the next host. The encapsulated offers consist of the hash function and then the encrypted offer which consist of the generated offer and random number by the visited hosts S₀,S₁,…,Sₙ. The protocol is discussed in below.

4.1 Agent at the Creator (S₀)

The creator S₀ creates and starts the agent with the dummy offer O₀ and random number r₀. And then signs the offer Sig₀(O₀) for Non-repudiability and encrypt those data Encᵢ₀(Sigᵢ₀(Oᵢ), rᵢ) with the help of the public key Pb₀ and selects the next host S₁. Also it generated the temporary public and private key (tP₀, tP₁) for sign in the identity of the next host Sig₁(tP₀).
The process of each host is depicted in the flowchart beginning.

Where reliable. This is by the public key the recipient or not and also the offer from the previous host is encapsulated offer for the integrity that is to check whether he is O. Agent migrates from

4.2 Agent at S1

Agent migrates from S0 to S1 and carries the encapsulated offer O0. After receiving the encapsulated offer, S1 can verify the encapsulated offer for the integrity that is to check whether he is the recipient or not and also the offer from the previous host is reliable. This is by the public key Pb0 of the sender.

S1: Receive O0
   Recover C0, h0 by Pb0
   Ver(Sig1(S0), Sig2(S1), tPb0) recover S1, S0

Where S1 (identity) is recovered by tPb0 and the S0 (identity) is recovered by Pb0 and verify the identity sequence from the beginning.

Generate offer O1
   Compute C1 = EncPub(Sig1(O1), r1)
   Generate tPr1, tPb1
   Decide next host S2
   h2 = H(Sig1(S0), Sig2(S1), Sig3(S2), tPb1, r0)
   O1 = Sig1(O1, h2)
   S1 → S2: O1, tPb1

4.3 Agent at S2

After receiving the encapsulated offers, host S2 recovers the encapsulated offers O1 by the help of the tPb1 and O0 by the help of Pb0. After that the identity of the two encapsulated offers O1, O0 are recovered. The recovered identities of O0 is S0, S1 and the identities of O1 is S1, S2, S3.

Now the last two identities of the O1 and first two identities of O1 is compared O0 (S0, S1) = O1 (S0, S1). If both are same the next process will proceeds otherwise it sends the agent back to its home.

S2: Receive O0, O1, tPb1
   Recover C0, h0 by tPb1
   Recover C0, h0 by Pb0
   Ver(Sig1(S0), Sig2(S1), Sig3(S2), tPb1, r0) recover S0, S1
   Ver(Sig1(S0), Sig2(S1), tPb0) recover S0, S1

After that it will generates its offer O1 and random number r1 and then computes encrypted C1 with the public key of creator Pb0.

4.4 Agent at S3

S3 receives the all previous encapsulated offers and verify the identities.

S3: Receive O0, O1, O2, O3, ..., O1, tPb1
   Recover C1, h1 by tPb1
   ...
   Recover C0, h0 by Pb0
   Ver(Sig1(S0), Sig2(S1), Sig3(S2), tPb0, Sig4(S3), tPb1, r0) recover S2, S3
   ...
   Ver(Sig1(S0), Sig2(S1), Sig3(S2), tPb0, tPb1, r0) recover S2, S3
   ...
   Ver(Sig1(S0), Sig2(S1), tPb0) recover S0, S1

It will recover the identities in the following format and compares the last two and first two identities. If both are same, the host decides no attack on that data else attack is identified and then the host sends the agent to its home. It will match from O0 to O1 for the reliability of the chain.
process, the host, which selected by the current host. Now the identity of the host identity represented in the diagonal is the next visiting host. Each host will have the three identities except the host O. Which is contained in the hash function. After the verification process, the host S generates its offer and made the hash function and forwards the agent to its next host S_{i+1}.

\[
\begin{align*}
\text{Generate offer } & O_i \\
\text{Compute } C_i = & Enc_{pr0}(Sig_{pr0}(O_i), r_i) \\
\text{Generate } & tPb_i, tPb_{i+1} \\
\text{Decide next host } & S_{i+1} \\
& h = H(Sig_{pr0}(S_{i+1}, tPb_{i+1}), Sig_{pr0}(S_i, tPb_{i+2}), \\
& \quad Sig_{pr0}(S_{i+2}, tPb_{i+2}), tPb_i, r_i) \\
& O_i = Sig_{pr0}(C_i, h) \\
S_i \rightarrow S_{i+1}: & O_0, O_1, O_2, \ldots, O_{i+1}, tPb_i
\end{align*}
\]

4.5 Agent return to home S_0

At last agent returns to its home and give the collected offers to its owner. The owner will recover all the encapsulated offers O_0, O_1, O_2, \ldots, O_i, O_{i+1}, \ldots, O_n with the help of the public key. Where O_0, is recovered by the tPb_0, and then O_{i+1} is recovered by the help of the tPb_{i+1}, which is already available in the hash function of O_0. After that the offer O_0, O_1, O_2, \ldots, O_i, O_{i+1}, \ldots, O_n are recovered by the public key of the originator Pb_{0}.

Fig.2. shows the agent migration from the owner to the other host to collect the offer and return back to the owner.

4.6 Security properties

Data Confidentiality: Only the originator can decrypt the offer O_i because it is encrypted by the public key Pb_{0} of the host S_0.

Non-repudiation: Each offer O_i is signed by its host S_i as Sig_{pr0}(O_i). So S_i cannot deny its offer O_i after S_i receives.

Forward Privacy: Even though the identities of the creators of offer O_i extracted by anyone but they are not able read or update the offer except the originator S_0. Because of the encryption mechanism Enc_{pr0}(Sig_{pr0}(O_i), r_i).

Strong Forward Integrity: None of the encapsulated offers can be modified because the each and every encapsulated offers includes the chain relation S_0, S_1, S_2 and S_n. If any result gets modified, the chain relation will not followed and then the current host identifies the attack and sends the agent to its home. Random number r_i also generated by the host for the purpose of the integrity. It is available in both the hash function and the encrypted function. The random number r_i in the both will be checked for the integrity. There may be chance to attack the part of the encapsulated offer. O_i = Sig_{pr0}(C_i, h). There is the possibility to change the C_i and make the new offer O_i = Sig_{pr0}(C_i’, h). Which will be identified during the decryption, with the help of r_i.

Insertion Resilience: No offer can be inserted in the middle and also no more than one offers can be inserted into the chain (Revisiting attack[8]) by the host, because the identities will repeat more than once in the chain. For example S_0, S_1, S_2 and S_n, now the host S_2 compares the chain and identifies the dual insertion by the single host with its identity. Same way the modification H(Sig_{pr0}(S_{i+1}, tPb_{i+1}), Sig_{pr0}(S_i, tPb_{i+2}), Sig_{pr0}(S_{i+2}, tPb_{i+2}), tPb_i, r_i) = H(Sig_{pr0}(S_{i+1}, tPb_{i+1}), Sig_{pr0}(S_i, tPb_{i+2}), \\
Sig_{pr0}(S_{i+2}, tPb_{i+2}), tPb_i, r_i) also identified because it violates the hash function.

Truncation Resilience: No offer O_i or sequence of offers from O_0, O_{i+1}, \ldots, O_n can be truncated and then the append the new offer. Because the comparison function is ended with the owner encapsulated offer O_0, so it can be easily identified by the following function.

\[
\begin{align*}
\text{Ver}( & Sig_{pr0}(S_0), Sig_{pr0}(S_1), Sig_{pr0}(S_2), tPb_0) \quad \text{recover } S_0, S_1, S_2 \\
\text{Ver}( & Sig_{pr0}(S_0), Sig_{pr0}(S_1), Sig_{pr0}(S_2), tPb_0) \quad \text{recover } S_0, S_1, S_2
\end{align*}
\]

Public Verifiable Forward Integrity: Any host can recover the encapsulated offer O_0, \ldots, O_i to verify the h by the help of the temporary public key generated by the previous host. The offer O_i = Sig_{pr0}(C_i, h) is recovered by the help of tPb_i which is received with the chain O_0, O_1, O_2, \ldots, O_{i+1}, O_{i} and tPb_i. Then
the \( Q \) contains the temporary public key for \( O_{i,2} \). Likewise it will recover the chain.

5. MULTIPLE COLLUDED TRUNCATION ATTACKS

Various mechanism is provided to defend the truncation attacks. Darren et al.[11], pointed to protect the two-colluder truncation attacks which will also avoid the attacks by the single host[2]. And also the they provide the protection for Multiple (three or more) colluded attacks while they are not in the adjacent place. But our proposed protocol, can protect the multiple colluded attackers when they are in the adjacent places also because the chain relation is maintained in the form of the previous, current and then the next host identities. In the chain \( S_1,\ldots,S_{m-1},S_m,S_{m+1},\ldots,S_{n} \), if \( S_i \) and \( S_n \) are colluded to attack the sequence of offers from \( O_1, O_2, \ldots, O_m \). Now the offers will be \( O_1, O_2, O_3, \ldots, O_{m-1}, O_{m+1}, \ldots, O_n \). Now this chain is forwarded to the host \( S_{m+2} \). This host will recover this offers and verify the identities as represented in the Fig.1.

6.GROWING A FAKE STEM ATTACK & INTERLEAVING ATTACK

Attacker truncates the offers \( Q \) and appends fake offers is referred as growing a fake stem attack[2]. Attackers breaks the chain and provide the space for inserting the fake offers is interleaving attack[10]. In this protocol, there is no opportunities for this type of attack because each will maintain the identity, by the identity the fake stem and Interleaving attack will easily identified as described in the Insertion resilience.

7.CONCLUSION

Our protocol uses the identity chain relation with the collected offer. So the free roaming agent will be protected from the two-colluded or multiple-colluded attacks. It can also avoid the Growing of fake stem attack[2]. Revisiting attack[8] and Interleaving attack[10]. In this method we can avoid the concentration of the previous host while we reach the next host in the previous protocol[11]. This method will avoid the network traffic and also very much useful in the Distributed environments (which will be used to help the free roaming mobile agent in the E-Commerce & E-learning environments to collect the results.

8.ACKNOWLEDGMENT

This Work is supported by the NTRO, Government of India. NTRO provides the fund for collaborative project “smart and secure environment” and this paper is modeled for this project. Authors would like to thanks the project coordinators and the NTRO members.

9.REFERENCES


