Programming Satan's Agents

Volker Roth
Fraunhofer Institute for Computer Graphics
Dept. Security Technology
Rundeturmstrasse 6
64283 Darmstadt

January 12, 2003

Abstract

Mobile agent security is still a young discipline and most naturally, the focus up to the time of writing was on inventing new cryptographic protocols for securing various aspects of mobile agents. However, past experience shows that protocols can be flawed, and flaws in protocols can remain unnoticed for a long period of time. The game of breaking and fixing protocols is a necessary evolutionary process that leads to a better understanding of the underlying problems and ultimately to more robust and secure systems. Although, to the best of our knowledge, little work has been published on breaking protocols for mobile agents, it is inconceivable that the multitude of protocols proposed so far are all flawless. As it turns out, the opposite is true. We identify flaws in protocols proposed by Corradi *et al.*, Karjoth *et al.*, and Karnik *et al.*, including protocols based on secure coprocessors. Additionally, we propose how the protocols can be strengthened against the types of attacks we launch against them.

Keywords: mobile agent security, breaking security protocols.

1 Introduction

This paper was inspired by a previous work by Ross Anderson and Roger Needham, titled "Programming Satan's Computer" [1], which was published in 1995. A version of its abstract serves us as introduction. We took the liberty to alter the original text slightly, so that it better fits the context of mobile agent security.

"Cryptographic protocols are used in systems for mobile agents to protect the confidentiality and integrity of data acquired by agents [are used in distributed systems to identify users and authenticate transactions]. They may involve about 2-5 protocol steps per hop [the exchange of 2-5 messages], and one might think that a program of this size would be fairly easy to get right. However, this is absolutely not the case; bugs can be found [bugs are routinely found] in known protocols, and years after they were first published. The problem is the presence of a hostile opponent, who can create agents and alter an agents' contents at will [alter messages at will]. In effect, our task

is to program mobile agents which give answers [a computer which gives answers] which are subtly and maliciously wrong at the most inconvenient possible moment. This is a fascinating problem; and we hope that the lessons learned from programming Satan's agents [computer] may be helpful in tackling the more common problem of programming Murphy's."

Analyzing cryptographic protocols for mobile agent protection means meeting old friends and foes. Some of these protocols are flawed in ways similar to those already pointed out by Anderson and Needham in the context of programming Satan's computer. We first summarize the typical objectives of the protocols we analyze:

Objective 1 (Confidentiality) *Mobile agents shall reveal cleartext only while being on trusted hosts.*

Objective 2 (Integrity) *The agents shall be protected such that they can acquire new data on each host they visit, but any tampering with pre-existing data must be detected by the agent's owner (and possibly by other hosts on the agent's itinerary).*

The general objective here is to protect certain features of a mobile agent against malicious hosts. By assumption, the host of the agent's owner is always trusted. Some of the protocols address both objectives simultaneously, others address just one. All protocols are targeted at protecting *free-roaming* mobile agents. In other words, mobile agents that are free to choose their respective next hop dynamically based on data they acquired in the course of their execution.

Our attacks follow two general patterns; variations and combinations of them are executed repeatedly:

cut & paste: The attacking host cuts a portion out of one agent, and pastes it into another agent. In effect, data taken from one protocol run is used in another.

oracle exploit: The attacking host generates an agent of its own and dispatches it to one or more remote hosts on which the agent is transformed as required by the cryptographic protocol.

The combination of both patterns enables attacks on cryptographic protocols devised in [7, 3, 5, 6]. In some cases this leads to a complete compromise of the protocol's security objectives. In other cases the adversary is able to forge and replace subsets of the protocol data in a way that makes it impossible for an agent's owner to detect the tampering. The important observation here is not that protocol data acquired by agents can be truncated (some authors already acknowledge this possibility) but that the attacker can exercise control over the data returned by an agent.

2 Some Protocol Failures

We will write encryption of some *plaintext* into a *ciphertext* symbolically as $c = \{m\}_K$, where K is the *key* being used. A digital signature will be written as an encryption with a private signing key S^{-1}. We will write $S^{-1}(m)$ when we refer to the bare signature rather than the union of the signature and the signed data. We assume

that the identity of the signer can be extracted from her signature. A cryptographic hash of some input will be written $h(m)$. Unless noted otherwise, we assume that h is *preimage resistant* and *collision resistant* [8, §9.2.2], which implies that h must also be *2nd-preimage resistant* [8, §9.2.5]. When A sends some message m to B we will write $A \rightarrow B : m$. We will write $A \rightarrow B : \{m\}_{K_{A,B}}$ when m is sent over a confidential channel. Concatenation of m_1 and m_2 is written as $m_1 \mid\mid m_2$. For ease of reading, we refer to some entities by their nicknames, e.g. Alice, Bob, and Eve. In general, Eve will play the role of the adversary, Alice will play the role of the victim agent's owner, Bob and Dave will play the role of additional entities taking part in the protocols. The itinerary of Alice's agent is written as $i_0, \ldots, i_n$, where $i_0 =$ Alice and i_n is the host currently visited by the agent.

2.1 Decrypting the targeted state

In [7], Karnik and Tripathi propose a *targeted state* as a means to protect the confidentiality of data carried by an agent. The idea is to make this data available to the agent only when it is on a host that is trusted with respect to keeping this data confidential from other agents and hosts. In order to achieve this, the plaintext is encrypted with the public key of the trusted host. The targeted state looks like this:

$$\{\{m_1\}_{K_{i_1}}, \ldots, \{m_n\}_{K_{i_n}}\}_{S_A^{-1}}$$

The targeted state is signed by Alice, who is the originator of the agent owning the targeted state. Having received an agent, each host inspects the targeted state for ciphertexts it can decrypt. If so, the host decrypts it using its own private decryption key, and makes the cleartext available to the agent.

Below, we illustrate the attack on this protocol. Without loss of generality, we assume that the agent's targeted state contains a single ciphertext, which is encrypted with the public key of Bob. Alice first sends the agent to Eve from whom it hops to Bob and then returns to Alice. The protocol starts as follows (for simplicity, we assume here that an agent initially consists only of its targeted state and its program Π_A):

$$A \rightarrow E \quad : \quad \Pi_A, \{\{m\}_{K_B}\}_{S_A^{-1}}$$

The attack is straightforward. Eve strips off Alice's signature, copies $\{m\}_{K_B}$ into the targeted state of an agent of her own, signs this targeted state, and sends her agent to Bob:

$$\begin{aligned} E \rightarrow B \quad &: \quad \Pi_E, \{\{m\}_{K_B}\}_{S_E^{-1}} \\ B \quad &: \quad \Pi_E, \{\{m\}_{K_B}\}_{S_E^{-1}}, \{\{m\}_{K_B}\}_{K_B^{-1}} = m \end{aligned}$$

Bob innocently decrypts the targeted state using his own private key and makes the resulting plaintext available to the agent. The agent then migrates back to Eve carrying the plaintext.

$$B \rightarrow E \quad : \quad \Pi_E, \{\{m\}_{K_B}\}_{S_E^{-1}}, m$$

Eve now is in possession of the plaintext which should be available only to Bob; Alice never detects the attack. The problem with this protocol is that, due to a lack of redundancy in the ciphertext, Bob can be abused as an oracle. Alice needs to include an unforgeable identifier of her agent in the ciphertext, e.g. $h(\Pi_A, A)$. Even then, the agent's program must be unique for each agent[1] and designed carefully such that it can not be abused in the way illustrated above by means of malicious state changes.

2.2 Forging the append only container

In addition to the *targeted state*, Karnik and Tripathi also propose an *append only container*. The idea is to protect a container of objects in an agent such that new objects can be added to it but any subsequent modification of an object contained therein can be detected by the agent's owner. The protocol relies on an encrypted checksum, whose initial value is computed by Alice (the agent's owner) based on a random nonce r as follows:

$$C_0 \quad = \quad \{r\}_{K_A}$$

The nonce must be kept secret by Alice, and is used in the verification protocol upon the agent's return. The append only container is defined as follows:

$$\{\{m_1\}_{S^{-1}_{i_1}}, \ldots, \{m_n\}_{S^{-1}_{i_n}}, C_n\}$$

Whenever a new object is appended to the append only container, the checksum is updated[2] as given below:

$$C_{n+1} \quad = \quad \{C_n \mid\mid S^{-1}_{i_{n+1}}(m_{n+1})\}_{K_A}$$

The signer of the appended object is the host on which the append operation takes place. Upon the agent's return, Alice successively decrypts the checksums, extracts the signature, and verifies the signature against the corresponding object in the container. The last checksum must equal the initial nonce.

We now assume that Eve received Alice's agent and she knows C_j for some $1 \leq j \leq n$. Eve always knows C_n, because it is embedded in the container. She might collude with other servers which the agent visited before, or she might be part of a loop in the agent's itinerary. In these cases, Eve might discover a checksum C_j with $j < n$.

At this stage, Eve has multiple choices. She can either truncate the container up to the j'th object and grow a fake stem by releasing the agent. Or she can remove, add or replace arbitrary objects m_l with $l > j$ in the name of other hosts. In order to do this, Eve creates an agent with the object that she wants to add at $j + 1$, and an append only container of her own, with checksum C_j as its initial value. Eve now sends her agent

[1] Otherwise Eve can still cut & paste targeted states back and forth between agents that are owned by Alice and which share the same program.

[2] In the original protocol description, the signature and identity of the server are appended. On the other hand, we assume that the signer's identity can be extracted from the signature and appending it is, therefore, redundant.

to Bob. There, Eve's agent inserts m_{j+1} in its own targeted state and migrates back:

$$\begin{aligned} E \to B &: \Pi_E,\ m_{j+1},\ \{C_j\} \\ B \to E &: \Pi_E,\ \{\{m_{j+1}\}_{S_B^{-1}}, \{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_E}\} \end{aligned}$$

Upon the agent's return, Eve decrypts the checksum using her own private key, and re-encrypts it using the public key of Alice:

$$\begin{aligned} C_{j+1} &= \{\{\{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_E}\}_{K_E^{-1}}\}_{K_A} \\ &= \{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_A} \end{aligned}$$

Then, she constructs a new container:

$$\{\underbrace{\{m_1\}_{S_{i_1}^{-1}}, \ldots, \{m_j\}_{S_{i_j}^{-1}}}_{\text{from A's agent}}, \overbrace{\{m_{j+1}\}_{S_B^{-1}}, C_{j+1}}^{\text{from E's agent}}\}$$

which replaces the previous container of Alice's agent. This process is repeated with the new checksum until Eve is pleased with the result, and releases Alice's agent. Bob is not able to detect the attack, because C_j is not publicly verifiable (it is encrypted with Alice's public key). All Bob can see is the length of C_j, from which he can estimate the number of objects that must be in the append only container. So if Eve wants to make sure that Bob has no reason to get suspicious then she adds j signed objects to her agent's container before she sends it to Bob. As long as these objects are properly signed it does not matter who signed them and where she got them.

Once again, a lack of redundancy allows Eve to abuse other hosts as oracles, this time for the purpose of signing and checksum computation rather than decryption.

2.3 Forging the multi-hops protocol

In [3], Corradi, Montanari, and Stefanelli propose a protocol they call *multi-hops*, which has the same purpose as the *append only container* presented by Karnik and Tripathi. It falls prey to the same type of attack. However, this time, the faked agent needs to do one more hop to complete its attack. For reference, we summarize the multi-hops protocol below.

Let $(\Pi, \mathcal{M}, \mathcal{P})$ be an agent where Π is static (immutable) code and initialization data, $\mathcal{M}$ is (mutable) application data, and $\mathcal{P}$ is protocol data (meta information required by the protocol). Alice initializes her agent with $(\Pi_A, \epsilon, \epsilon)$. The protocol additionally requires a nonce γ and a message authentication code μ. The initial values are $\gamma_0 = h(r)$ and $\mu_0 = \epsilon$, where r is chosen randomly. On each hop, the agent can add some data M to its application data, which is then protected by the host using the multi-hops protocol. The protocol is defined as given below:

$$\begin{aligned} \gamma_n &= h(\gamma_{n-1}) \\ \mu_n &= h(m_n, \gamma_{n-1}, \mu_{n-1}, i_{n+1}) \end{aligned}$$

$$\begin{aligned} \mathcal{P}_n &= \mathcal{P}_{n-1} \mid\mid S_{i_n}^{-1}(\mu_n) \\ \mathcal{M}_n &= \mathcal{M}_{n-1} \mid\mid m_n \mid\mid i_n \end{aligned}$$

$$i_n \rightarrow i_{n+1} \quad : \quad (\Pi_A, \mathcal{M}_n, \mathcal{P}_n), \{\gamma_n\}_{K_{i_{n+1}}}, \mu_n$$

The message authentication code γ_n serves as a *chaining relation* that binds results previously obtained by the agent to the ones obtained at the current host and to the identity of the agent's next hop.

Due to this chaining relation, the attack cannot be executed in the same way as it is done for the append only container. The resulting star shaped itinerary with Eve in the center would be too obvious in the protocol data. What Eve has to do here is to plan ahead one step.

Again, we assume that Eve knows some γ_{j-1} for $1 \leq j \leq n$. She received the agent so she always knows γ_{n-1} and μ_{n-1}. She can obtain γ_{j-1} with $j < n$ by colluding with other hosts or as a result of loops in the agent's itinerary. Let i_{j+1} be Bob, and i_{j+2} be Dave. Eve now sets $\mathcal{M}_j = \mathcal{M}_{j-1}$, $\mathcal{P}_j = \mathcal{P}_{j-1}$, $\gamma_j = \gamma_{j-1}$, $\mu_j = \mu_{j-1}$, and does the following:

$$\begin{aligned} E \rightarrow B \quad &: \quad (\Pi_E, \mathcal{M}_j, \mathcal{P}_j), \{\gamma_j\}_{K_B}, \mu_j \\ B \rightarrow D \quad &: \quad (\Pi_E,\ \mathcal{M}_j \mid\mid m_{j+1} \mid\mid B,\ \mathcal{P}_j \mid\mid S_B^{-1}(\mu_{j+1})), \\ &\qquad \{\gamma_{j+1}\}_{K_D}, \mu_{j+1} \\ D \rightarrow E \quad &: \quad (\Pi_E,\ \mathcal{M}_j \mid\mid m_{j+1} \mid\mid B \mid\mid M^* \mid\mid D, \\ &\qquad \mathcal{P}_j \mid\mid S_B^{-1}(\mu_{j+1}) \mid\mid S_D^{-1}(\mu_{j+2})), \\ &\qquad \{\gamma_{j+2}\}_{K_E}, \mu_{j+2} \end{aligned}$$

Eve sends her agent first to Bob where it inserts some m_{j+1} chosen by her. Then it hops to Dave (who is the next target), inserts some random data M^* (which is discarded later on), and returns to Eve. Eve now updates Alice's agent as shown below, using the data acquired by her own agent:

$$\begin{aligned} & (\underbrace{\Pi_A,\ \mathcal{M}_j}_{\text{A}} \mid\mid \overbrace{m_{j+1} \mid\mid B}^{\text{E}},\ \underbrace{\mathcal{P}_j}_{\text{A}} \mid\mid \overbrace{S_B^{-1}(\mu_{j+1}))}^{\text{E}} \\ = \quad & (\Pi_A, \mathcal{M}_{j+1}, \mathcal{P}_{j+1}) \end{aligned}$$

Eve now plans her next move (whom she wants to attack after Dave). In order to send the agent to Dave she needs to know γ_{j+1} and μ_{j+1}, but she doesn't – yet. However, Eve knows γ_j, μ_j, and m_{j+1}. This is sufficient to compute

$$\begin{aligned} \gamma_{j+1} &= h(\gamma_j) \\ \mu_{j+1} &= h(m_{j+1}, \gamma_j, \mu_j, D) \end{aligned}$$

At this stage, Eve either continues the attack, or it releases Alice's agent and sends it to Dave, where it resumes normal execution.

$$E \rightarrow D \quad : \quad (\Pi_A, \mathcal{M}_{j+1}, \mathcal{P}_{j+1}), \{\gamma_{j+1}\}_{K_D}, \mu_{j+1}$$

The underlying weakness of the multi hops protocol is the same as in the previously described protocols, namely, the abuse of servers as oracles.

3 The KAG family of protocols

Karjoth, Asokan, and Gülcü [5] published a family of protocols which are directed at preserving the integrity and confidentiality of data acquired by free-roaming agents. The general scenario is that of a comparison shopping agent that visits a number of shops, and collects offers from them.

3.1 Publicly verifiable chained digital signatures

The *Publicly verifiable chained digital signature protocol* (P1) is defined as given below:

$$\begin{aligned}
\mathcal{M}_n &= \{\{m_n, r_n\}_{K_A}, C_n\}_{S_{i_n}^{-1}} \\
C_n &= h(\mathcal{M}_{n-1}, i_{n+1}) \\
\mathcal{M}_0 &= \{\{m_0, r_0\}_{K_A}, C_0\}_{S_A^{-1}} \\
C_0 &= h(r_0, i_1) \\
\\
i_n \rightarrow i_{n+1} &: \Pi, \{\mathcal{M}_0, \ldots, \mathcal{M}_n\}
\end{aligned}$$

where m_0 is a dummy offer, r_n is random salt that makes it harder to attack the encryption. C_n is called the *chaining relation* at n. By assumption, it shall be possible to deduce the identity of the signer from a signature [5, pp. 198]. The signer of $\mathcal{M}_0$ is deemed to be the owner of the agent (unfortunately, the authors of [5] do not explicitly mention from what they conclude who the owner of a given agent is, so we have to do a little guesswork here).

The security of the protocol relies on the assumption that an attacker does not change the last element $\mathcal{M}_n$ in the chain. However, there is no reason why an attacker would be so obliging. On the contrary, if the attacker is willing to build a complete chain for the agent then he can even remove chain elements *before* his own entry (this contrasts with e.g. the *honest prefix* property introduced by Yee [11, pp. 267]). The important observation here is that the input to all previous chaining relations is known.

We assume that Eve received an agent owned by Alice. Let Eve be $i_n, n > 1$. She picks j with $0 < j < n$ and a new i_{j+1} of her choice. Please note that there is no free choice of i_j once j is fixed, only of i_{j+1}. Eve has to collect an offer from the original shop i_j for her chosen j in order to maintain the chaining relation's validity at $j - 1$. Then Eve does the following:

$$\begin{aligned}
E \rightarrow i_j &: \Pi_E, \{\mathcal{M}_0, \ldots, \mathcal{M}_{j-1}\} \\
i_j \rightarrow i_{j+1} &: \Pi_E, \{\mathcal{M}_0, \ldots, \mathcal{M}_j\} \\
i_{j+1} \rightarrow E &: \Pi_E, \{\mathcal{M}_0, \ldots, \mathcal{M}_{j+1}\}
\end{aligned}$$

Upon the agent's return, Eve throws away $\mathcal{M}_{j+1}$, increments j, and picks a new i_{j+1}. The chaining relation and encapsulated offers are build as if Alice's agent had requested the offer (instead of Eve's agent with Eve's *program*) because $\mathcal{M}_0$ bears Alice's signature. Eve repeats the process at her discretion. When she is finally satisfied with the collection of encapsulated offers she assembled, she pastes them into Alice's agent, and sends that agent to i_{j+1}. If Alice can be fooled into forwarding agents whose $\mathcal{M}_0$ she signed herself then Eve's charade can carry on until the very last (faked) hop. Otherwise, Eve has to stop her attack before the next to last hop.

It must be stressed here that the problem is *not* that Eve can truncate offers and grow a fake stem (this possibility is acknowledged by the authors, so this fact is not surprising). The problem is that shops can be abused as oracles for generating offers to the terms of Eve rather than Alice (this remark also holds for sections 3.2 and 3.3). In other words, Alice might look for blue or red shirts with a preference on blue ones; she might find out that Eve is the only shop that offers her blue shirts, though. This is possible because Eve's agent looks only for red shirts, and the offers made to this agent are returned to Alice.

3.2 Chained digital signatures with forward privacy

The second protocol proposed in [5] is the *chained digital signature protocol with forward privacy*. It is a variation of the protocol discussed in section 3.1, with the order of encryption and signature computation being swapped. The goal of this arrangement is to hide the identity of shops that provided offers while keeping the integrity assurances. The protocol is defined as given below:

$$\begin{aligned} \mathcal{M}_n &= \{\{m_n\}_{S_{i_n}^{-1}}, r_n\}_{K_A}, C_n \\ C_n &= h(\mathcal{M}_{n-1}, r_n, i_{n+1}) \end{aligned}$$

$$i_n \rightarrow i_{n+1} \quad : \quad \Pi, \{\mathcal{M}_0, \ldots, \mathcal{M}_n\}$$

A problem we couldn't resolve is how a shop knows who the owner of an agent is, and hence for whom the offers must be encrypted. The shop cannot extract the identity of Alice from $\mathcal{M}_0$, because the signature of the dummy offer m_0 is hidden by the encipherment. The authors leave that to speculation. The protocol's description is far from being sufficiently detailed at this point. Whereas a signer's identity can be verified easily against her signature using a public key and corresponding certificate (where the identity binding is assured by a certification authority), anybody could have used somebody else's *public* key to encrypt data.

Again, we assume that Eve received Alice's agent, and Eve is i_n as in the previous attacks. Let j be the smallest number for which Eve knows i_j. Eve probably knows i_{n-1} because this is most certainly the host that sent her the agent. In any case she knows i_n (her own identity).

Eve collects arbitrary signed offers using agents of her own, including an offer from i_j. Then, she cuts off the chain at j, and appends the offers, starting with the fresh one collected from i_j and the remaining ones in arbitrary order. In doing so, she generates random nonces as required, and builds the chaining relations consecutively

from known data. The last chaining relation is computed with the identity of the entity to whom Eve wants to hand off Alice's agent.

Upon the agent's return, Alice cannot decide whether her agent remained unattacked, or carries offers of shops it has never seen actually. It is worth noting that the integrity assurance of the protocol relies on the secrecy of the association of $\mathcal{M}_j$ with the identity of the shop who signed offer m_j. This means that privacy of offers is not only a *feature* of the protocol, but is also a *requirement*. In particular, secrecy of the agent's itinerary is a requirement.

Once again, not the truncation of protocol data is the important point, but Eve's ability to set the terms for (authentic) offers returned to Alice.

3.3 Publicly verifiable chained signatures

The remaining protocol proposed in [5] is the *publicly verifiable chained signatures* protocol. The key aspect of the protocol is that each shop generates a temporary asymmetric key pair (either on the fly or by means of pre-computation) to be used by the successor. The public key is certified by the shop that generated the key pair. Each shop uses the private key that it received from its predecessor for signing its partial result, the chaining relation, and the public key to be used by its successor. The private key is destroyed subsequently. Let (χ_A, χ_A^{-1}) be a temporary key pair generated by A. The protocol is as follows:

$$\begin{aligned} \mathcal{M}_n &= \{\{m_n, r_n\}_{K_A}, C_n, \overbrace{\chi_{i_n}}^{\text{oracle}}\}_{\chi_{i_{n-1}}^{-1}} \\ C_n &= h(\mathcal{M}_{n-1}, i_{n+1}) \\ i_n \rightarrow i_{n+1} &: \Pi, \{\mathcal{M}_0, \ldots, \mathcal{M}_n\}, \underbrace{\{\chi_{i_n}^{-1}\}_{K_{i_n,i_{n+1}}}}_{\text{oracle}} \end{aligned}$$

The protocol is initialized by Alice with:

$$\begin{aligned} \mathcal{M}_0 &= \{\{m_0, r_0\}_{K_A}, C_0, \chi_A\}_{S_A^{-1}} \\ C_0 &= h(r_0, i_1) \end{aligned}$$

It is easy to see that Eve can collect valid certified temporary key pairs from Bob, simply by dispatching and agent of her own to Bob, which promptly returns to Eve. On the agent's transport to Eve, Bob sends a temporary private key χ_B^{-1} and corresponding certified public key χ_B (contained in $\mathcal{M}$).

We assume that Eve is i_n and she received Alice's agent. Let j be the smallest number for which Eve knows $\chi_{i_j}^{-1}$. She received $\chi_{i_{n-1}}^{-1}$ with the agent, so at least one such j exists and $j < n$. Eve then cuts off all encapsulated offers following $\mathcal{M}_j$, and collects key pairs from all the shops in whose names she wants to fake offers, including shop i_{j+1}. Starting with i_{j+1}, she appends arbitrary offers, building the protocol data consecutively. The identity that Eve uses in the final chaining relation is the one of the entity to whom she wants to hand off Alice's agent (for instance Alice herself).

4 Protocols using secure coprocessors

In [6], Karjoth proposes use of trusted secure coprocessors as a means to protect mobile agents in a distributed marketplace. The setting equals that described in section 3, with the exception that each shop i_n has a trusted tamperproof hardware T_n (in brief, its *device*), which is issued and certified by a central market authority $\Re$. The market authority acts as a trusted third party for merchands and customers. By assumption, the channel between a shop and its device is secure against active attacks. Each device has its own asymmetric key pair, and is capable of computing suitable asymmetric ciphers, symmetric ciphers, and message digests. Furthermore, each device has the public key of the market authority, and uses it to authenticate the public keys of other devices.

At the beginning of the protocol, Alice chooses a random $\mathcal{K}$ and sets $C_1 = h(\mathcal{K})$. The protocol continues as follows:

$$\begin{aligned}
i_{n-1} \rightarrow i_n &: \Pi_A, \{\mathcal{K}, C_n\}_{K_{T_n}}, \{\mathcal{M}_1, \ldots, \mathcal{M}_{n-1}\}, \\
&\quad \{C_1, \ldots, C_{n-1}\} \\
i_n \rightarrow T_n &: \{\mathcal{K}, C_n\}_{K_{T_n}}, \{m_n\}_{S_{i_n}^{-1}}, \{K_{T_{n+1}}\}_{S_{\Re}^{-1}} \\
T_n &: \mathcal{M}_n = \{\{m_n\}_{S_{i_n}^{-1}}\}_{\mathcal{K}},\ C_{n+1} = h(\mathcal{M}_n, C_n) \\
T_n \rightarrow i_n &: \{\mathcal{K}, C_{n+1}\}_{K_{T_{n+1}}}, \{C_{n+1}\}_{\mathcal{K}}, C_n, \mathcal{M}_n \\
i_n \rightarrow i_{n+1} &: \Pi_A, \{\mathcal{K}, C_{n+1}\}_{K_{T_{n+1}}}, \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \\
&\quad \{C_1, \ldots, C_n\}
\end{aligned}$$

In the final protocol step, the last shop sends Alice the agent and the final checksum, which is encrypted with $\mathcal{K}$:

$$i_n \rightarrow i_0 \quad : \quad \Pi_A, \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \{C_1, \ldots, C_n\}, \{C_{n+1}\}_{\mathcal{K}}$$

Alice knows $\mathcal{K}$, so she decrypts $\{C_{n+1}\}_{\mathcal{K}}$, verifies the checksums consecutively from C_1 to C_{n+1}, decrypts $\mathcal{M}_1, \ldots, \mathcal{M}_n$, and finally she verifies the signatures.

We assume that Eve runs a shop in the electronic marketplace, which implies that she has a device certified by the market authority. Consider that Eve received an agent owned by Alice, so Eve is i_n. Eve now has a number of encrypted offers, an equal number of checksums, and $\{\mathcal{K}, C_n\}_{K_{T_n}}$, which can be decrypted only by her device.

From the protocol, we know that $C_{n+1} = h(\mathcal{M}_n, C_n)$. There is nothing secret about h, so in fact Eve can take j of the $n-1$ encrypted offers, shuffle them, and re-compute the appropriate checksums herself, beginning with the initial checksum C_1 (without ever going through her device). However, Alice expects to receive a matching $\{C_{j+1}\}_{\mathcal{K}}$ with her agent. Eve cannot encrypt her final checksum with $\mathcal{K}$ because she does not know it – but her device can do it for her! All Eve has to do is passing C_{j+1} in the place where her device expects to receive Eve's signed offer:

$$E \rightarrow T_n \quad : \quad \{\mathcal{K}, C_n\}_{K_{T_n}}, \underbrace{C_{j+1}}_{\text{substituted for Eve's offer}}, \{K_{T_n}\}_{S_{\Re}^{-1}}$$

The device first extracts Alice's secret key $\mathcal{K}$ from $\{\mathcal{K}, C_n\}_{K_{T_n}}$, which is encrypted with the device's public key. Then the device uses $\mathcal{K}$ to encrypt what it thinks is Eve's signed offer. Only that it is not the signed offer but the checksum that must be passed back to Alice with her agent.

$$T_n \quad : \quad \mathcal{M}_n = \underbrace{\{C_{j+1}\}_{\mathcal{K}}}_{\text{oracle computation}},\ C' = h(\{C_{j+1}\}_{\mathcal{K}}, C_{j+1})$$

Eve also passed her own device's public key rather than that of another shop's device. What Eve gets back from her device is:

$$T_n \rightarrow E \quad : \quad \{\mathcal{K}, C_{n+1}\}_{K_{T_n}}, \{C'\}_{\mathcal{K}}, C_n, \underbrace{\{C_{j+1}\}_{\mathcal{K}}}_{\text{leaked result}}$$

In other words, given a set of signed offers $\mathcal{M}_1, \ldots, \mathcal{M}_j$ (which are encrypted with Alice's secret key $\mathcal{K}$), Eve can construct a valid representation of Alice's agent, and return it to Alice in a way that is indistinguishable from an ordinary run of the agent.

Eve can also collect signed offers herself (at her own terms) using agents of her own. For instance, let $\{m_B\}_{S_B^{-1}}$ be such an offer, collected from Bob. Eve sends this offer to her device, rather than one of her own offers:

$$E \rightarrow T_n \quad : \quad \{\mathcal{K}, C_n\}_{K_{T_n}}, \underbrace{\{m_B\}_{S_B^{-1}}}_{\text{Bob's offer}}, \{K_{T_n}\}_{S_{\Re}^{-1}}$$

$$T_n \rightarrow i_n \quad : \quad \{\mathcal{K}, C_{n+1}\}_{K_{T_n}}, \{C_{n+1}\}_{\mathcal{K}}, C_n, \underbrace{\mathcal{M}_B}_{\text{Bob's offer encrypted with } \mathcal{K}}$$

The device returns the offer encrypted with $\mathcal{K}$. Offers prepared in this way can also be used by Eve in her attack on the checksum.

If Eve just wants to append offers that she collected to Alice's agent (following $\mathcal{M}_{n-1}$), then the attack is even simpler. All Eve has to do is passing her own device's public key to her device rather than that of another shop's device until she wants to hand off Alice's agent. In that case she either passes the public key of the next shop's device, or returns the agent to Alice herself.

In summary, Eve can delete and rearrange any offers brought by the agent, and insert forged offers collected by her, at any position[3] in the chain of results. This means in particular that the protocol does not achieve *forward integrity* as is claimed by its author. The surprising fact is that although secure coprocessors are used, the protocol fails where some software only approaches succeed (for instance the *chained MAC protocol* [5]). The lesson that is to be learned is that tamperproof hardware is no guarantee for improved security.

[3] In general, Eve knows only $\{\mathcal{K}, C_n\}_{K_{T_n}}$, so if she touches any encrypted offers before n then she has to hand off the agent herself to Alice, and cannot let another shop do this. However, she can pass on the agent if she knows that it will return to her before it hops back to Alice.

5 How can we be saved?

We described the crypto protocols design problem for mobile agent security as "programming Satan's agents" because a mobile agent under the control of an adversary is possibly the most obstructive piece of code which one could send. It may give answers which are subtly and maliciously wrong at the most inconvenient possible moment. In order to be saved, we must check first that a protocol does not commit the old familiar sins, because the Devil did not have to come up with a new and pernicious twist in the attacks we described. In 1995, Anderson and Needham gave some rules of thumb of good and bad practice; we quote three of them below:

- "be clear about the purpose of encryption — secrecy, authenticity, binding, or producing pseudorandom numbers. Do not assume that its use is synonymous with security;"
- "where the identity of a principal is essential to the meaning of a message, it should be mentioned explicitly in that message."
- "be careful, especially when signing or decrypting data, not to let yourself be used as an oracle by the opponent;"

We might continue and quote entire sections. Their plea remains valid as ever. Instead, we concentrate on particularities of mobile agents. A mobile agent is comparable to an active "token" that is passed around by the entities who participate in a protocol. An important entity – the agent's owner – participates in general only at the beginning and end of a lengthy protocol run (which is part of the demand for *autonomy* of agents). Agents can be underway for some while, therefore it is difficult to have a notion of "freshness". Any timeout must probably be so long that an attack can likely be carried out within the validity period of the protocol data. Furthermore, the *identity assumption* does not hold for mobile agents, which is a cornerstone of many "traditional" security protocols. Chess [2] describes the identity assumption as follows:

> "Whenever a program attempts some action, we can easily identify a person to whom that action can be attributed, and it is safe to assume that that person intends the action to be taken."

This assumption fails when applied to mobile agents, because "a migrating agent can become malicious by virtue of its state getting corrupted" [4]. We cannot assume that an agent properly represents the intentions of its owner, because – subsequent to its first hop – an agent's state is a function of its own program and state, and the state and program of the hosts that it visited.

Let us assume, for a moment, that the identity assumption held for mobile agents. When Alice sends an agent, can we use Alice's identity then, and follow the second rule? Certainly not, because Eve can still cut & paste protocol data among two different agent *instances* owned by Alice. Different agents represent different protocol runs, and – again – Anderson and Needham already noted that we must "be sure to distinguish different protocol runs from each other." This leads to the important conclusion that

digitally signing a mobile agent's code alone is not sufficient to assert agent ownership.

However, this approach is a favorable one among contemporary mobile agent systems. A signature on code can be copied just like the code itself. Code is written to be re-used, so the agent *instance* is what renders an agent (a protocol run) distinct. Seen in this light, it is even less desirable to sign credentials that contain a *code base* rather than the code itself (as described e.g. in [7]), because this gives Eve potentially more agent programs to choose from. Each agent program that is available from a particlar code base can be used in conjunction with credentials that refer to the code base.

Preferably, Alice signs a static *kernel* of her agent (which must of course include the code required by this particular agent). The kernel must be unique for each of her agents. Protocol data must be bound to this kernel and Alice's identity. This does not relieve us from the burden to write agent programs which cannot easily be abused by means of malicious state changes, but serves as an anchor for protocol data, and makes it harder to abuse legitimate hosts as oracles. We give a sketch of how this might work in section 6.

6 Repairing protocols

One problem repeatedly occurred in the protocols we analyzed: a legitimate host could be abused as an oracle that decrypts, signs, or otherwise computes protocol data on behalf of an adversary. Mobile agent systems are particularly vulnerable by this type of attack because they are meant to work autonomously, and no human intervention is expected to happen in order to validate and authorize the execution of cryptographic services provided to agents. Hence, agent servers and agent owners must have means to decide whether protocol data that an agent requests to process or returns, actually belongs to that agent. Below, we sketch two approaches suitable to solve this problem.

The first approach binds confidential data to a particular agent instance so that this binding can be verified by the decryptor before decryption takes place. It is a slightly simplified version of a protocol we describe in [10].

The second approach demonstrates how data, that is acquired by mobile agents dynamically, can be authenticated and bound to a particular agent instance so that Alice can immediately spot attempts to fake or reuse protocol data.

6.1 Binding confidential data

The idea is to construct an agent kernel and ciphertexts in a way that allows authorized hosts to detect whether a ciphertext brought by an agent actually belongs to the agent. For simplicity, we assume that there shall be only one ciphertext and one authorized host. Let Bob be an authorized host. Alice prepares her agent as follows:

$$\Phi_{r_0} = \underbrace{\{\{\Pi_A, r_0, h(S_A, K_0)\}_{S_A^{-1}}}_{\text{The agent's } \textit{kernel}}, \{m_0\}_{K_0}, B, \{K_0\}_{K_B}\}$$

where r_0 is a random number and K_0 is a randomly generated secret key, both long enough to make any chance of being used twice (in the case of r_0) or found by exhaustion attacks in due time (in the case of K_0) negligible, and S_A is Alice's public signing key (more precisely, her signing key's public key certificate). We assume that Alice's identity can be derived from her signature. Signature verification shall imply verification of the identity to public key binding, e.g. by means of public key certificates using a trusted public key infrastructure. Alice dispatches her agent which eventually migrates to Bob: $A \xrightarrow{*} B : \Phi_{r_0}$

Bob verifies Alice's signature on the agent's kernel. If the verification succeeds then Bob recovers K_0 from $\{K_0\}_{K_B}$ using his private decryption key. Next, he computes $h(S_A, K_0)$ and verifies the result with its counterpart in the agent's kernel. If both match then Bob decrypts $\{m_0\}_{K_0}$, and makes m_0 available to the agent.

The value of $h(S_A, K_0)$ serves as a proof that the alleged owner of $\{m_0\}_{K_0}$ actually *knows* the secret decryption key K_0. The proof can be verified only by entities who can recover K_0, and it cannot be modified without breaking the digital signature of the agent that owns the encrypted data.

The effect is that Eve, having received Φ_{r_0}, cannot use an agent of her own to plant an egg and have it hatched by Bob. Neither can she plant her egg in another instance of Alice's agent, because each kernel is unique. Eve has to tamper with Φ_{r_0} in a way such that it goes to Bob, takes m_0, stores the plaintext in a place where Eve can get to it later on, and returns to her. As a consequence, Π_A must be designed carefully to cope with this threat.

6.2 Binding acquired data

In this section we construct an agent kernel Φ, and we derive an *unique implicit name* ϕ from it, in brief *implicit name:*

$$\Phi_{z_A} = \underbrace{\{\Pi_A, z_A\}_{S_A^{-1}}}_{\text{the agent's kernel}} \qquad \phi_{z_A} = \underbrace{h(S_A^{-1}(\Pi_A, z_A))}_{\text{the implicit name}}$$

where z_A is a random number large enough to make any chance of being used twice by Alice negligible. Whenever an agent with kernel Φ_{z_A} requests Bob to sign some protocol data m, Bob actually computes $\{m, \phi_{z_A}\}_{S_B^{-1}}$. In other words, m is valid only in the context of the agent instance whose implicit name is signed along with m. This binding can be verified by Alice upon her agent's return. As a consequence, protocol data that was not requested by Alice's agent is rejected.

Please note that z_A is required and must not be re-used with the same agent program Π_A. For practical purposes, the current time should suffice (we anticipate that attacks on Alice's clock are infeasible).

7 Conclusions

> *The greatest trick the Devil ever pulled was convincing the world he didn't exist.*
>
> — from motion picture "The Usual Suspects"

Judging from the number of respective publications we could find by means of the World Wide Web, research in basic cryptographic protocols for protecting mobile agents seems to have petered out after 1998 when compared to the flurry of the two years before. One reason might be that there were protocols around that seemed to solve a fair amount of what is considered achievable in software-only approaches. In this paper, we would like to reinvigorate this area of research.

We analyzed seven protocols which have the objective to protect secrecy and integrity of data carried or acquired by mobile agents from malicious hosts. In all cases, the protocols turned out to be vulnerable to a combination of cut & paste attacks and oracle exploits. These vulnerabilities stem from the fact that certain design rules set forth already years ago were not followed faithfully.

Additionally we proposed improvements to the analyzed protocols in order to render them more robust against the types of attacks we launched against them. In particlar, we described how mobile agent kernels must be designed that uniquely identify a particular instance of a mobile agent, and showed how such kernels can be used e.g. to prevent abuse of legitimate hosts as oracles for decrypting and signing data brought and acquired by mobile agents.

Despite the improvements we proposed, software-only protection against general truncation attacks is still out of reach. Assume that Eve knows Alice's agent in state σ_j. Later, Eve receives the same agent in state σ_n with $n > j$. Eve may always reset the agent to state σ_j unless there is either a notion of freshness (tricky, we discussed this briefly in section 5) or an *external state* not under the control of Eve (which is at the bottomline of what we proposed in [9]).

8 Acknowledgments

We thank Günter Karjoth for his comments on the initial manuscript, which helped to improve the paper's precision and focus, as well as the inspiring discussion that evolved from the commentary.

References

[1] Ross Anderson and Roger Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441. Springer Verlag, 1995.

[2] David M. Chess. Security issues in mobile code systems. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 1–14. Springer Verlag, Berlin Heidelberg, 1998.

[3] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents protection in the Internet environment. In *The 23rd Annual International Computer Software and Applications Conference (COMPSAC '99)*, pages 80–85, 1999.

[4] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for mobile agents: Issues and requirements. In *Proceedings of the National Information Systems Security Conference (NISSC 96)*, pages 591–597, October 1996.

[5] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free–roaming agents. In K. Rothermel and F. Hohl, editors, *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, volume 1477 of *Lecture Notes in Computer Science*, pages 195–207. Springer Verlag, Berlin Heidelberg, September 1998.

[6] Günter Karjoth. Secure mobile agent-based merchant brokering in distributed marketplaces. In D. Kotz and F. Mattern, editors, *Proc. ASA/MA 2000*, volume 1882 of *Lecture Notes in Computer Science*, pages 44–56. Springer Verlag, Berlin Heidelberg, 2000.

[7] Neeran M. Karnik and Anand R. Tripathi. Security in the Ajanta mobile agent system. Technical Report TR-5-99, University of Minnesota, Minneapolis, MN 55455, U. S. A., May 1999.

[8] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and its Applications. CRC Press, New York, 1996. ISBN 0-8493-8523-7.

[9] Volker Roth. Mutual protection of co–operating agents. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 275–285. Springer-Verlag Inc., New York, NY, USA, 1999.

[10] Volker Roth and Vania Conan. Encrypting Java Archives and its application to mobile agent security. In Frank Dignum and Carles Sierra, editors, *Agent Mediated Electronic Commerce: A European Perspective*, volume 1991 of *Lecture Notes in Artifical Intelligence*, pages 232–244. Springer Verlag, Berlin, 2001.

[11] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 261–273. Springer-Verlag Inc., New York, NY, USA, 1999.