Integrating Java-based Mobile Agents into Web Servers under Security Concerns*

Stefan Fünfrocken
Department of Computer Science, Darmstadt University of Technology
Alexanderstr. 6, 64283 Darmstadt, Germany
Email: fuenf@informatik.tu-darmstadt.de

Abstract

The paper describes a system architecture which offers the ability to host mobile agents (so-called Web-agents) on a Web server. This is done by a special server extension module called 'server agent environment' (SAE). The agents may access local data of the Web server and may communicate with other Web-agents or with human users. The paper discusses the different security issues that arise in such a system and shows how we address the problems. Concerning system and network security, we present a solution based on security packages, protection domains, and agent capabilities. This provides a flexible way to restrict an agent's possibility to access the local server data or access the network. Since we also aim at providing our SAE as a plug-in for other Web servers, we show how this is supported by our system architecture.

1 Introduction

The notion of 'mobile agents' commonly refers to programs that are able to move from host to host. Since the term 'agent' often implies some sort of intelligent behavior and most developers in the mobile agent area are not primarily concerned with artificial intelligence issues, the paradigm is also known as 'mobile code' or 'remote programming'. On every host the agent is visiting, it may access local data or may communicate with other agents to fulfill its task.

The mobile agents paradigm was first promoted by General Magic [35]. Although the idea of migrating processes is not new - there exist operating systems that use process migration to balance the load of a network of computers – the vision of an electronic world of itinerant agents which fulfill tasks on behalf of their users attracted quite some attention. Because of the rather restrictive information policy of General Magic concerning implementation details of their system, several research groups began to develop mobile agent systems (i.e., systems that enable the execution of agents at a host) on their own. There have been several mobile agent systems [12, 31] before Sun released the object serialization mechanism as part of the remote method invocation package of the Java language in 1996 [1]. But since object serialization offers a very easy way of implementing migrating objects, there are more and more mobile agent systems that are programmed in Java and use Java to program the agents [20] - even General Magic stopped the Telescript development and is now distributing an agent system based on Java [10]. Unfortunately, Java is not designed as agent system programming language – as was Telescript – and therefore most agent related functionality has to be added by additional libraries.

Because the insertion of foreign code into a local host is in the heart of the mobile agent paradigm, security of a mobile agent system is the most important concern for developers of such a system as well as for site administrators installing it. A mobile agent system should be an execution environment for foreign agents which a local user could trust. This trust should be motivated either because the mobile agent system is under local control of the site administrator or because the agent system's implementation meets certain safety and security constraints and offers specific safety and security properties at the application level.

The most immediately recognized security property of mobile agent systems is the ability to protect the resources of the local host from foreign agents, because mobile agents may be compared, in some sense, to computer viruses. But there a two other important security aspects: protection of an agent from other

agents and from malicious hosts is the most important property for agent programmers and agent users – especially in scenarios, where agents carry precious data which eventually may lead to some loss of real world values, when misused.

Typically, early mobile code systems [12, 17, 25, 31] were developed by using interpreted languages as an agent programming language and imposing a security policy on 'dangerous commands' of the language, which led to so-called 'safe' versions of the language interpreters, thus providing effective means to protect the underlying host from any program executing in the 'safe' environment.

Farmer et.al. [7] show that mobile agent security is hard to implement in general, but there are well-known techniques to ensure a certain level of security for protecting hosts from agents or for protecting agents from each other. In contrast to this, protecting an agent from any malicious or faulty host is very hard and even impossible with respect to certain aspects. This stems from the fact that in the first scenario the malicious object (the agent) is under control of the entity which is to be protected (the local host). In the second scenario, the entity to protect (the agent) is given away to be under control of foreign objects (the remote host or agent system).

This motivates why security is most important for mobile agent systems. Our paper presents the security aspects of our Java-based architecture which enables agents to migrate between World Wide Web servers. Section 2 gives a short overview of our system architecture (see [9] for a more complete description). Section 3 describes mobile agent safety and security issues. Section 4 deals with safety and security aspects of Java, which we use as an agent programming language. Section 5 and Section 6 show the solution of our implementation in more detail. Finally, Section 7 is about which additional security issues we have to solve if we want to provide our SAE as a plug-in for other servers.

2 WASP System Architecture

The infrastructure we present was developed as part of the WASP project. With the '**W**eb **A**gent-based **S**ervice **P**roviding' project, we aim at providing services on Web data and using mobile agents to implement these services. The underlying hypothesis is that services for the World Wide Web is one application domain for which the mobile agent paradigm is a well-suited model.

Our system architecture consists of an HTTP server which offers standard Web server functionality, and a special server extension module called 'Server Agent Environment (SAE)' which provides the mobile agent environment for that server. As other HTTP servers, our server provides full Web server functionality, in serving local Web data to remote users by file content or by executing cgi-bin scripts and serving their output. In addition to that, our server is able to host mobile agents, so-called Web-agents. A Web-agent's code may be installed at some Web server, which then is the home server of any agent generated from this code. Thus, our server has to handle the generation, execution, and migration of agents. All these agent related tasks are forwarded to the server's agent environment, which takes care of them (see Figure 1). Starting an agent means to send an HTTP-get request

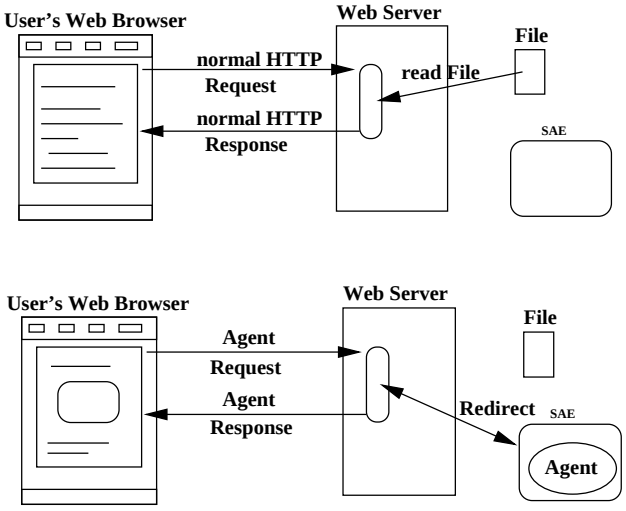

Figure 1: General WASP Infrastructure

to a Web-agent related URL. The actual start of the agent is done by the server's SAE. After being loaded and initialized by the SAE, the agent may send its GUI to the user. This is done by answering the get request with a generated HTML page, which consists of an applet representing the agent's GUI. Web-agents initiate their migration by calling a 'go' method. The actual (network) transfer of agents in migration is realized with an HTTP-post request to a SAE specific URL at the target Web server. The body of the post request consists of the agent coded into a multipart MIME message [3].

3 On Safety and Security

When talking about safety and security of a hardware and software system, the specific meaning of both terms should be explained. We will use the terms safety and security in the context of programming languages and software systems according to the following specification (see also [5, 11]).

Security deals with the rules for the interaction of different objects and is concerned about integrity and protection. There has to be a security policy – either an implicit or an explicit one – which is enforced by the software system. Informally, the security policy regulates 'who is allowed to do what' in a system, this includes object access and manipulation. When implementing a security system, the security policy requires the existence of specific system mechanisms which realize the policy: For example, in a mobile agent system where access to system resources like secondary storage is granted to specific objects within certain limits, there has to be a mechanism to trace the amount of secondary storage consumption for each object. Since security policies may change over time, the security implementation should be flexible or parameterizable.

In the context of programming languages, **safety** means that a component is 'safe to use'. This can be seen as an assurance that the component behaves as specified, thus describing an operational property of a component. There have to be safety constraints (implying hazard assessment [22]) which define the 'correct' behavior of components. A safe system is designed and (hopefully) implemented in a way that it always remains in 'permitted' system states. More generally, safety is concerned about making efforts to reduce 'the risk of harm (to persons) or damage' [27].

A mobile agent system is, as any software system or application, not a monolithic component: it is rather a combined, layered, or hierarchical system of components. Depending on the life-cycle of an application, the application has to *meet* the predefined safety and security requirements (while being developed), or *guarantees* certain safety and security properties to any user of the application. Ideally, required and guaranteed properties should match[1].

Note that there can be no security without safety: Any component enforcing a security policy uses mechanisms which provide specific functionality, like identifying the object accessing a resource. If the mechanism's implementation is not safe, there is no guaranty that the identification returned is the correct one.

The safety and security properties of a mobile agent system or one of its subcomponents can be divided into several categories (see also [11]), each owning specific mechanisms to implement or achieve the properties. Each of the categories deals with different aspects of the application or component, thus requiring specific properties of the implementation or the way an application is constructed from subcomponents. Often, it is not easy (and sometimes impossible) to distinguish between requirements that lead to safety properties of an application and requirements that lead to security properties: there are requirements that lead to both. Also, the different categories are not orthogonal in the sense that a mechanism ensuring a safety property in one category may be used to ensure a security property in another one. The categories are the runtime category, the object interaction category, the underlying system category and the network category (see Figure 2) which directly relate to the mobile agent system security issues (see also [19]):

- agent to host security (underlying system),
- agent to agent security (object interaction),
- host to host security (network), and
- host to agent security (underlying system, object interaction).

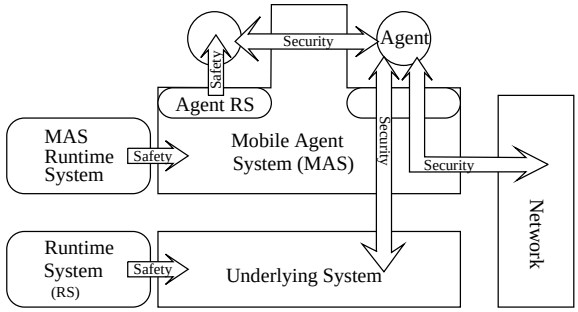

Figure 2: Safety and Security in Mobile Agent Systems

As mentioned before, there can be no security without safety. Because of this, the runtime system the mobile agent system is running on has to provide runtime safety so that the mobile agent system can implement its security mechanisms on top of it. As a mobile agent system is the runtime system for the mobile agents, it has to provide runtime safety to the

[1] The verification of this requires a formal description of the properties of the application and formal program verification. This is a difficult task and therefore seldomly done for today's software. See [24, 28, 29] for first steps in this direction concerning the Java Bytecode.

agent, so the agent may make use of some application level security mechanisms. Since mobile agent systems may use and combine low level security mechanisms of the underlying system to realize an internal security mechanism, the runtime system of the underlying system has to provide runtime safety too. Typical mechanisms to ensure runtime safety are: memory protection, memory management, runtime type checking, array-bounds checking, exceptions, and exception catching.

In each of the categories mentioned above, a component exhibits safety and security properties. Normally the properties of subcomponents are combined when building a mobile agent system to provide some application-level safety and security properties to the users of the system. While there are good solutions for the first two security issues, there exist currently only first steps for solutions for the last two issues [7, 15, 32].

4 Java — Safety and Security

Our system is based on Java. We use Java as an implementation language for our HTTP server and the server's SAE. Furthermore, agents in our system are also programmed in Java. Because of this, our system inherits Java's safety and security properties in the categories *runtime* and *object interaction*.

Not much can be found about Java's runtime safety [5, 18, 29], but since Java provides concepts such as strong typing, no address arithmetics, array bound-checking, and exception handling, it is presumed to be safe although there are some weaknesses in Java's type system [8]. Most Java related security information [30] is about security of applets, which can be viewed as simple agents that can be (down-) loaded on demand. Applets are run in a so-called Java Sandbox, which can be viewed as a very simple 'agent environment', where applets are allowed to do almost nothing. This is controlled by the Java SecurityManager class. This should leave no room for any misbehaving applet. However, there are many known hostile applets which exploit weaknesses in Java's *implementation* of safety and security[2]. This also proves that Java cannot serve as a mobile agent system by itself, just because it supports remote execution of code and provides a simple sandbox model.

One major problem in solving these Java-related problems is that they would require a modification of the Java Virtual Machine (VM), which is beyond the scope of almost any mobile agent system developer group. There are currently two mobile agent systems [25, 26] that use a modified Java VM, but the modification was necessary to support transparent agent migration for Java-based agents and was not done to improve the security or safety of Java.

For our system we rely on the safety and security properties of Java, since we are implementing our security architecture on top of the security architecture implementation of Java. So any flaw in that implementation will introduce a security problem in systems using Java. Even with the new and more flexible Java security model [13], there remain many security problems a mobile agent system has to solve by providing it's own security architecture.

There is more ongoing research to improve Java's security properties [28], in particular in the mobile code scenario [33].

5 HTTP Server Security

Basically HTTP servers make the data contained in local files available to remote users. Remote users access the server and the server's data through the HTTP protocol, which provides different requests for different actions on the server's data. The most widely used request is the HTTP-get request, which offers read-only access to the requested document. Many servers don't even implement other HTTP requests as for example put or delete, which write to a file and delete a file, respectively.

Since there exists data which should not be showed to everybody, most Web servers offer the possibility to restrict the access to certain files to users who have to identify themselves and must be authorized to view the data contained in the requested file. Such a protected file set is called 'realm' or 'protection domain'. The realm consists of the list of users and their passwords, which may access the files protected by the realm.

Security of the server not only means to protect the local data from unauthorized access. This can easily be established by the means of realms. It also means to protect the local system (i.e., guarantee system security). There exist quite some security problems when allowing the execution of cgi-bin programs. These local programs are executed when users request a cgi-bin URL. Normally, these programs generate HTML pages as output, thus offering dynamic Web pages. Since the cgi interface is able to pass parameters to the program, remote users can exploit safety and se-

[2] This shows that it is not easy to prove that an implementation of a mechanism that should enforce security meets its specification. See [6] for a taxonomy of Java bugs.

curity weaknesses of the programming language the program is written in. This can lead to unallowed access to the system the Web server is running on. Since the programming language of cgi-bin programs is beyond the control of the server, any server allowing cgi-bin programs is prune to this type of security weakness. This is also true for our Web server, since we support execution of cgi-bin programs. A solution to this problem is the new server side scripting possibility of servlets, which are Java programs that behave like cgi-bin programs, but offer the safety and security features of the Java language.

The security architecture of our HTTP server is based on the idea that there is a managing component for each system resource, which stores and enforces the security configuration reflecting the security policy for that specific resource. Currently there exist three components: first a component that knows about the identity of registered users, the *id-manager*, which stores the user name and the Web server password of the user; a component that manages the capabilities of any registered user, the *capability-manager*, which stores what rights each user is granted by the Web-server administrator; and a component that knows about the realms that are defined, the *realm-manager*, which stores the file list, access rights, and user list for each realm. Consider an incoming HTTP request

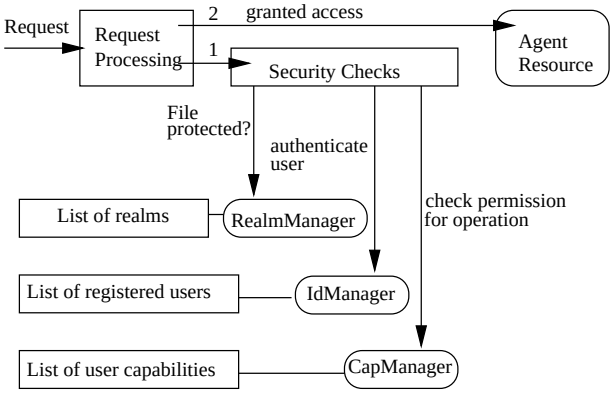

Figure 3: Security checks for any http access

to start an agent (see Figure 3). First it is checked with the realm-manager if the URL pointing to the agent is protected by a realm indicating that only the users that are registered in the realm are allowed to start the agent. After that, it is checked if the request carries a user supplied user name and password. If so, it is checked whether the user is a registered user and can be authenticated with the password. Before checking that the authenticated user is a realm member it is made sure by asking the capability manager if the user is allowed (specified by the server administrator) to start agents at the local server. If the request passes all checks the request is fulfilled by instructing the server's SAE to start the agent.

In our system, each realm may define read, write, delete, and execute rights for the files protected by the realm in two categories: for the users in a privileged list, and for all other users. Users who can identify themselves to the server by presenting their user name and password and who can be authorized for the realm by being part of the realm's privileged user list, are granted the privileged rights of the realm. A failed identification or authorization will grant only unprivileged rights to the user. The realm-manager may allow registered users to define new realms, but only on their own files. As a consequence, each realm is owned by a registered user. The owner may also grant special rights to some other registered users, allowing them to modify the access rights to the realm or the list of privileged users[3]. The server administrator can register users with the id-manager, and may grant or deny several rights, as for example the right to create own realms. As explained in the next section, there are also several other rights that the server administrator can grant to registered users.

6 SAE Security

The 'Server Agent Environment' is a mobile agent system designed to allow Web servers the hosting of a special kind of agents, so-called Web-agents. Web agents are started at some server where the agent was installed either by the server administrator or by some user to which the server administrator granted the right to install agents. Once started, the agent – which can be seen as the local representative of the user who started the agent – may access the Web server's local data. Because of this, the SAE has to guarantee that the agent can access only those local Web server data which the user who started the agent could access. For users accessing the Web server with their Web browsers, this access restriction is enforced by the Web server according to a protection scheme. Thus, the SAE has to respect and enforce the same protection scheme for agents as the Web server is using for normal users. To ensure this, our SAE uses the same security architecture as our Web server and makes use

[3]This can, of course, introduce a security problem on a level which is beyond server control. A realm owner should grant those rights only to trusted users.

of the server's security components. To prevent any direct access to the server components from within the SAE, the shared objects have proxy representations inside the SAE which identify anyone accessing them before granting the access. Using proxy objects also facilitates the use of our SAE as plug-in. The proxy objects are used as mediator between the SAE objects and the foreign Web server's representations of for example the realm manager.

Since there may be more than one agent visiting the local Web server, we impose a scheduling on all currently runnable agents. This is done by the agent scheduler of the SAE, which grants time slices to each agent. Since there is only one agent running at any instant of time[4], we know which agent issued an resource access request. Since this is important to guarantee security (see section 6.3), the agent scheduler is part of the security architecture.

6.1 Agent Capabilities

Each agent is associated with a set of capabilities which describe the local rights the agent is granted, similar to Telescript permits [35] or Ara allowances [25]. These rights are computed locally, by combining the default system capabilities for the agent defined by the server administrator, the capabilities the user who started the agent (the agent user) granted to the agent, and the capabilities the user who installed the agent (the agent owner) granted to the agent. Depending on whether the agent is registered with the server or not, the default server capabilities are computed from the capabilities the administrator registered for that agent or from the capabilities for unknown agents. The resulting set of capabilities form the capabilities local to the current server. Since the default server capabilities for agents may differ from server to server, the resulting set differs too.

Our system uses two capability sets, one for the agent user and one for the agent owner. When an agent is started by another user and not by its owner, the agent is granted the capabilities specified by its owner in addition to the capabilities granted by the user who started the agent. Since this is a security compromising feature, this is not the default behaviour: the owner must switch on this feature explicitly. Agent owner and agent user can grant only equal or less powerful capabilities to the agent as they are granted themselves at the current server. Therefore, at least the owner or the user has to be registered at the current server. If none of the two is registered, the capabilities are ignored and the agent is granted the rights of anonymous agents. Of course, the resulting set is eventually restricted by the capabilities the local server administrator is granting. The list of capabilities one can grant to an agent includes:

- life time in CPU seconds
- allowed maximum secondary storage
- right to create, delete or modify realms
- right to create persistent local data
- right to start other local agents
- right to connect to the network
- right to access local resources directly

We do not control nor measure the main memory the agent is using, as the Java VM does not provide this information on a per thread group basis. Unfortunately, this threatens agent to host protection concerning denial of service attacks.

6.2 Agent Security Packs

When an agent migrates to a Web server's SAE, the SAE has to set up the local resource managers (id-manager, capability-manager, realm-manager) according to the user ids and capabilities under which the agent should be running. To handle this information setup, we developed so-called security packs. For each resource on which a security policy is imposed, there is a local resource manager and an associated security pack which knows how to set up the manager to achieve a setting of the manager configuration which complies to the local security policy (see Figure 4). Any agent access is – unnoticeable for the agent – intercepted by a security manager (see 6.3) which checks for the validity of the access before granting it. Currently there is an id-security-pack, a capability-security-pack, and a file-security-pack. A network-security-pack will get added soon. The packs set up a view on the local resources. Each security pack holds specific security related information: The id-security-pack carries the user ids and the corresponding passwords the agent is using to identify itself at each server. The capability-security-pack stores the capabilities that should be granted to the agent on different hosts. The file-security-pack knows about the files the agent should be granted access to. The files have to be

[4] We represent agents as separate threads inside the SAE and allow agents to create subthreads. Therefore we are scheduling thread groups of agents. So far, our system does not provide migration of the threads created by an agent, but there is currently ongoing research on that topic.

owned by a user present in the id-security-pack. The network-security-pack will hold information about the network accesses the agent should be granted. This in-

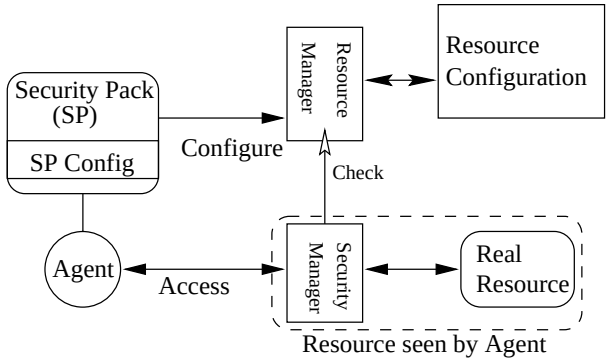

Figure 4: Security packs configure resource managers

formation, i.e. the security pack configuration, has to be carried along when an agent migrates from server to server: one user or owner could have different user ids and passwords on different servers or may grant access to different sets of files on different servers to the agent. Note, that only the security pack configuration is carried along and not the security pack itself. It is always a local security pack which sets up the local resource managers.

The security packs are configured after a user starts an agent and when the agent is configured through its GUI. When the agent is configured, the agent is in its configuration phase, and it is not permitted to do anything before this phase is terminated by the user pressing an 'ok' button, which is not part of the agents GUI. Besides the agent's GUI, the configuration GUI of the security packs is presented to the user. Both GUIs are contained in the WASP system GUI, which actually encapsulates the other two. The security packs are presented with default setup parameters. These defaults result from an intersection of the site defaults for each security pack and the defaults specified by the owner and programmer of the agent. The user may only further restrict these parameters according to his or her demands. In addition the user may impose his or her own security policy on the resources owned by him or her by specifying appropriate values for the security pack configuration (e.g., restricting the access to the files owned by the user). Before the agent can be 'set on the loose' by pressing the 'ok' button of the WASP system GUI, the user has at least to provide the id-security-pack with its user name and password, thus authorizing all other user related settings of the security packs'[5]. These settings are then attached to the agent and are migrated with it.

Since the data contained in the security packs is very sensitive, we encrypt all communication between the GUI parts that are loaded into the user's browser and the SAE. This is currently done by generating a session key and using the IDEA algorithm. In this way, the configuration information that is exchanged between the agent and its GUI is encrypted too.

The agent has no knowledge nor access to its attached security packs. Each security pack knows which SAE component to contact in order to set up that component according to the parameters carried. Migrated security pack configurations are not trusted data. The file-security-pack for example will only install file accesses which are authorized by the owner of the files to be accessed. Thus an agent is granted at the maximum the same access rights as the combined access rights of the users listed in the id-security-pack (intersected with the site restrictions).

Normally, the agent carries only those security packs it will need: an agent that will not access any file gets no file-security-pack attached, thus resulting in a local file view which contains no files: any access to a file will result in some 'file not found' exception. Currently, the agent programmer has to specify which security packs have to be attached to the agent. In the future we plan to provide an agent assembly tool which extracts this information from the agent's source code by inspecting it.

6.3 Agent to Host Security

The most obvious security property a mobile agent system has to provide is realized in our system by intercepting any method call of a Web-agent that access local resources and identifying who issued the access request. This can easily be achieved by installing an instance of an object that conforms to the Java Virtual Machine (VM) SecurityManager interface, which – when installed – is called by all Java methods that access system resources.

In our system (and in all other Java-based systems [10, 14, 21, 34]) the code of the mobile agent system and the code of agents is loaded into the same Java VM. Because of that, the installed security manager cannot automatically deny any access to local resources, because the mobile agent system itself has to access the local resources. To solve this problem,

[5]The password is used to identify the agent at the server. With forthcoming Java-APIs such as SmartCard API and access to local devices from within a Web browser, one could use a Smartcard to authorize the agent.

we use a security architecture as depicted in Figure 5. First the installed Java VM SecurityManager instance checks for the origin of the access[6]. Any access originating from the Web server classes is an access of the Web server implementation and is granted. This can be done, because the server has checked with its id- and realm-managers whether to deny or grant the access to the user initiating the request. When the call originates from the SAE part of our system, the call is forwarded to the SAE security manager. This managers decides whether the call resulted from an agent or the SAE implementation. Any SAE calls are granted directly. In contrast to this, agent calls entail the identification of the agent that issued the call, by asking the agent scheduler. Once identified, the SAE security manager checks with the id-manager, the capability-manager, and possibly the realm-manager whether it can grant the access to the agent.

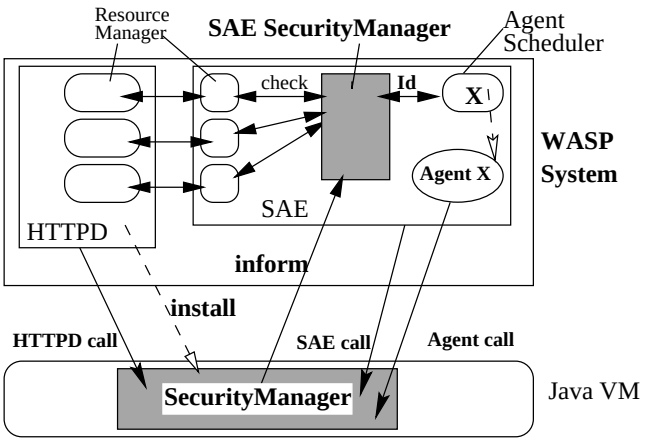

Figure 5: Security Architecture

6.4 Agent to Agent Security

Agents should not be able to harm each other in any way. In an object orientated implementation, agents are objects and may interact by retrieving a reference on each other and invoking methods on the other object. A mobile agent system that has to care for the security of such interactions, has to trace or restrict these interactions in some way, as for example was done by Telescript by introducing protected (i.e., read only) object references. Since Java does not provide any restrictions on object references, a Java-based mobile agent system has to monitor any object interaction (e.g., by using system proxies for each agent). Exchange of object references introduces some security and safety problems: when migrating an agent which holds a reference to some other agent using Java's object serialization, the second agent is serialized too. In that way, one agent can force another agent to migrate, which should not be possible in any way. Because of this, we currently provide no means for agents to retrieve a reference to any other agent. Instead of this, we provide communication streams between agents. An agent can request a communication stream to some other agent from the communication manager. What agents exchange through that channel is completely up to the agent programmers: agents may even decide to send references of themselves to each other by generating an object stream from the channel, but then the agent programmers have to be aware of possible problems and should mark the property holding the reference as 'transient'.

[6]This is done by inspecting the call stack. It relies on the Java VM's safety and security properties, which guarantee that no object may modify the Java VM stack.

6.5 Host to Agent and Host to Host Security

Achieving host to agent security is ambitious and rather difficult for the general case [7]. Therefore we designed our system as a net of trusted hosts as long as there are no satisfying solutions for this problem, thus defining away that problem to a certain extend. We plan to certify each SAE when delivered, and to give each SAE a signed public key. When migrating agents, a SAE requests the certified public key of the target SAE and encrypts each agent to be migrated with that key. In this way, it is ensured that agents are protected while traveling on the network and that SAEs allow agents to migrate only to certified SAEs. Thus we guarantee to each agent a trusted execution environment on a host. Of course, it is up to the agent user to trust the implementation of our security architecture. Note, that using SSL to transmit the post request migrating the agent encrypted, without encrypting the agent in the request body would compromise host to host security. The target Web server would receive the (SSL-) decrypted request which contains the unencrypted agent and a malicious Web server could modify the agent before handing it to its SAE. Because of this, the request body containing the agent has to be encrypted in a way that only the target SAE can read the agent. Thus using SSL is not necessary in our system, but could be used to hide even the type of the messages.

Concerning agent to host security in the general case, there is ongoing research, which seems to lead

to some partial solutions: see for example [15] which proposes agent code obfuscation and limited agent life cycles to prevent agents from being spyed out, or [32] which proposes an add on security architecture based on the concept of distributed transaction processing realised with CORBA and its object service for transactions (OTS).

7 SAE Plug-in Security Issues

One goal of our system architecture is to offer our SAE as a plug-in for other servers. To make use of our SAE, a foreign Web server has to provide an interface which allows to hand the incoming request to the SAE and which allows to hand back any response. Most existing Web servers offer the cgi-bin interface, which provides this. Java-based Web servers such as Jigsaw [2] or Jeeves [16] can integrate our SAE directly. We developed a Jigsaw resource object which connects our SAE to the Jigsaw resource tree and we are planing a Jeeves SAE servlet.

Connecting the SAE to other servers raises some security issues:

1. The SAE has to enforce the servers security policy for agents.
2. The SAE has to protect agents from the server.
3. Protection of the SAE from the Server and vice versa.

While we provide a solution for the first issue and partly for the second one, we currently see no solution for the third one.

Since Web agents executing in the SAE access local data which should be protected by the protection scheme a Web server is using, the SAE has to respect that scheme. Most existing Web servers offer simpler protection domains than our server is offering: on the one hand, they limit *any* access to the protected domain to the users registered in the realm. On the other hand, they do not intersect the rights when there is more than one realm defined for a file and a user is present in some of them. In addition to that, our scheme has a finer granularity by providing read, write, and execute rights for privileged and unprivileged accesses. Because of that, we have to incorporate the realm definitions of other servers into our realm system. In our system, agents are allowed to create and modify realms using our fine grain realm system. This implies that a user may not access data with a Web browser, whereas the SAE may grant the access for an agent started by the same user. We consider this not to be security problem, because the Web server would also grant the access of the user if it could handle the more flexible and fine grain scheme. The way we have to incorporate the realms depends on the way the SAE is connected to the server: when using the SAE as cgi-bin, we have no other choice than reading the server's realm definition file. This also means that any changes made to the realms have to be written to that configuration file. In contrast to that, Java-based Web servers can offer the possibility to access their realm objects. In this way, we can make use of that objects.

Adding our SAE to a Web server enables the migration of Web-agents to that server by using the HTTP-post request (i.e., the agent has to pass the server). Without any encryption of the agent the server would be able to modify the agent or the data the agent is carrying. To prevent that, the agent is transmitted in such a way that only the target SAE can read the agent by using public key encryption. Although this prevents the server from unrecognized tampering of the agent, a malicious server could modify the encrypted agent which would result in a failed agent migration. This is propagated to the origin SAE by the HTTP-post response which carries the information about the result of the migration. To prevent the unrecognized modification of that result, the target SAE has to sign it. By this way, we can detect but not prevent a security problem. Connecting our SAE to Java-based Web servers raises also an architectural problem which influences our security architecture: Java allows only a single instance of a security manager loaded into the Java VM. Our SAE does not install a security manager of its own (this is done by our Web sever implementation), but the SAE security manager has to be notified by the installed security manager when it detects a call resulting from within the SAE. We are currently working on that problem.

When our SAE is getting installed at some server it is not under our control any more. This is the same situation as for an agent traveling to a remote host. Therefore, protecting the SAE from the server is currently an unsolved problem. On the other hand, protecting the Web server form a malicious SAE is also hard, since the server has no knowledge of the SAE and therefore is not designed to protect itself. So far the only solution is trusting each other. The availability of trusted computing bases (TCB) would be a solution to this kind of problems.

8 Summary

In this paper we presented an architecture of a Web server enabled to host mobile agents by the means of a server extension module we call SAE. The system's security architecture provides safe and secure agent access to the underlying system resources and to the network. Agents are owned by an owner and started by a user, where each of them can define certain access restrictions for the agent by defining capabilities. The capabilities are carried along by the means of security package configurations which are not under agent control. These packages install a restricted view on the system resources for an agent before it interacts with the system. The agents resulting view is computed from server default security pack values which represent the server security policy and from user defined values which represent the users security policy for the agent. Using different security pack configurations allows a flexible way to define security policies. Our security architecture, which is based on cascaded security managers and agent capabilities, ensures that the agent cannot access anything which is beyond its viewing horizon, enforcing the system security policy for the agent.

References

[1] Arnold K., Gosling J., *The Java Programming Language*, Addison-Wesley, 1996

[2] Baird-Smith A., *Jigsaw Java HTTP Server*, by World Wide Web Consortium, http://www.w3.org/pub/WWW/Jigsaw

[3] Borenstein N., Freed N., *MIME (Multipurpose Internet Mail Extensions)*, Network Working Group, RFC1521, 1993

[4] Borenstein N., *EMail with a Mind of Its Own: The Safe-Tcl Language for Enabled Mail*, IFIP Transactions Comm. Syst., 1994, pp 389-402

[5] Dagenais M.R., *Building Distributed OO Applications: Modula-3 Objects at Work*, Draft Version, January 1997

[6] Dean D., Felten E. W., Wallach D. S., *Java Security: From HotJava to Netscape and Beyond*, Proc. of 1996 IEEE Symp. on Security and Privacy, May 1996, pp 190-200.

[7] Farmer W.M., Guttmann J.D., Swarup V., *Security for Mobile Agents: Issues and Requirements*, Proc. of NISSC96, 1996

[8] Fischbach R., *Java: Programmiersprache der Zukunft* (in German), iX 10/96

[9] Fünfrocken S., *How to Integrate Mobile Agents into Web Servers*, to appear in: Proc. of Wetice97 Workshop on CADWA, MIT, Cambridge, MA, June 1997

[10] General Magic, *Odyssey online information*, http:// www.genmagic.com/ agents/ odyssey.html

[11] General Magic, *An Introduction to Safety and Security in Telescript*, part of the Telescript documentation

[12] Gray R.S., *Agent Tcl: A flexible and secure mobile-agent system*, Proc. of the 4th Annual Tcl/Tk Workshop, Monteray, CA, 1996, pp9-23

[13] Gong L., *New security architectural directions for Java*, Proc. of IEEE COMPCON'97, Feb. 1997

[14] Hohl F., *Konzeption eines einfachen Agentensystems und Implementation eines Prototyps*, Diploma Thesis, Univ. of Stuttgart, Dept. of CS, Diplomarbeit Nr. 1267 (1995)

[15] Hohl F., *An approach to solve the problem of malicious hosts*, Univ. of Stuttgart, Dept. of CS, Fakultätsbericht Nr. 1997/03, (submitted to SOSP'97)

[16] Jeeves Team, *Overview of the Java HTTP Server Architecture*, Part of the Jeeves Alpha2 distribution, Sun Microsystems, 1996

[17] Johanson D., van Renesse R., Schneider F., *An Introduction to the TACOMA Distributed System*, Univ. of Tromsø, Dept. of CS, CS TR 95-23, June 1995

[18] Joyner I., *C++?? A Critique of C++ and Programming Language Trends of the 1990's*, 3rd Edition, October 1996

[19] Karjoth G., Lange B.D., Oshima M., *A Security Model for Aglets*, IEEE Internet Computing, July-August 1997, pp 68-77

[20] Kiniry J., Zimmermann D., *A Hands-On Look at Java Mobile Agents*, IEEE Internet Computing, July-August 1997, pp 21-30

[21] Lange D., Chang D.T., *IBM Aglets Workbench – Programming Mobile Agents in Java*, White Paper, IBM Corporation, Japan, August 1996,

[22] Leveson N.G., *Software Safety*, Communications of the ACM, Vol 34, No 2, Feb 91, pp 34-46

[23] Lingnau A., Drobnik O., Dömel P., *An HTTP-based Infrastructure for Mobile Agents*, WWW Journal - 4th Intern . WWW Conf. Proc. , Boston, MA, Dec 11-14, 1995

[24] Necula G.C., Lee P., *Safe Kernel Extensions Without Run-Time Checking*, Proc. of OSDI'96, Seattle, Washington, October 28-31, 1996

[25] Peine H., Stolpmann T., *The Architecture of the Ara Platform for Mobile Agents*, Proc. of MA'97, Berlin, April 7-8, LNCS 1219, pp 50-61

[26] Ranganathan M., Anurag A., Shamik S., Saltz J., *Network-aware Mobile Programs*, to appear in the Proc. of USENIX97

[27] Reliable Software Technologies, *Software System Safety Glossary*, http://www.rstcorp.com/safety-glossary.html

[28] Sirer E.G., McDirmid S., Bredshad B., *Kimera: A Java System Security Architecture*, http://kimera.cs.washington.edu/

[29] Stata R., Abadi M., *A type system for Java bytecode subroutines*, Digital Equipment Corporation, System Research Center, to appear in Proc. of SOSP'97

[30] Sun Microsystems, *Java Security online information*, http://java.sun.com/security/

[31] Thomsen B., Leth L., Knabe F., Chevalier P.-Y., *Mobile Agents*, European Computer-Industry Research Center (ECRC), report no. ECRC-95-21, 1995

[32] Vogler H., Moschgath M.-L., Kunkelmann T., *An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions*, Darmstadt Univ. of Technology, ITO, to appear in the Proc. of ICPADS'97

[33] Wallach D.S., Balfanz D., Dean D., Felten E. W., *Extensible Security Architectures for Java*, TR 546-97, Dept. of CS, Princeton Univ., April 1997

[34] Weiyi L., Messerschmitt D., *Java-To-Go, Itinerative Computing Using Java*, Univ. of California at Berkeley, Dept. of EE and CS, http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[35] White J.E., *Telescript Technology: The Foundation for the Electronic Marketplace*, Whitepaper by General Magic, Inc, Sunnyvale, CA, USA