Special Issue on Active Networks and Services, Computer Networks Journal, Elsevier, Volume 36, Issue 1, pp. 87-100, June 2001 (ISSN 1389-1286)

Computer Networks 36 (2001) 87–100

Security implications of implementing active network infrastructures using agent technology

Stamatis Karnouskos *

German National Research Center for Information Technology, Research Institute for Open Communication Systems (GMD-FOKUS), Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany

Abstract

Active networks (AN) are a rapid evolving area of research and in parallel an area of great industry interest. However, for this technology to make the step out of the labs and penetrate the market, the security problems have to be tackled effectively. This paper demonstrates why and how agent technology research, can and should be applied to active networks, in order to fulfill the new security challenges this infrastructure poses. First, we identify the key elements of AN, analyze the nature of active code, specify the role of agents in active networks and present a multi-execution environment active network architecture. Then, we target the security threats for active code and execution environment, and state the basic as well as the extended security requirements. Subsequently, we try to see how we can apply the security solutions and research done for agents to the context of active networks in order to satisfy their requirements.

Keywords: Active networks; Security; Active code; Agent technology

1. Introduction

Over the years, computer systems have evolved from monolithic and centralized computing devices to client-server environments that allow complex forms of distributed computing. Active networks (AN) [45] are an evolution of current dumb passive network carriers, where the level of abstraction is the protocol, to a more general programmable network model where the level of abstraction is raised to APIs for programming the new network resources. The idea is to move service code, which traditionally was placed outside the transport network, directly in network's nodes. Those nodes allow applications to configure them optimally for their tasks via open interfaces (programmable networks). Furthermore, those nodes will be able to compute on data they receive before they pass them to the next node (active networks). Network-aware software is expected to change the way we design and deploy applications and services. Dynamic Quality of Service, Quality of their Information and optimal exploitation for task specific computations will flourish within the AN community. The challenge such an infrastructure poses is to find the right balance among flexibility, performance, robustness, usability and last but not least security. Sophisticated security challenges have to be tackled effectively if this technology is ever to leave the research domain, penetrate the commercial sector and become widespread. In this direction, we are convinced that the agent technology can play a significant role.

* Corresponding author.
E-mail address: karnouskos@fokus.gmd.de (S. Karnouskos).

1389-1286/01/$ - see front matter
PII: S1389-1286(01)00155-4

Software agents [3] is a rapidly multi-developing area of research since the early 1990s. Agents can be classified by their characteristics such as mobility, intelligence, etc. For the active network case, we are especially interested in mobility but also partially in intelligence. Mobile agent technology offers a new computing paradigm in which a program in the form of a software agent (intelligent or dumb) can suspend its execution on a host computer, transport itself to another agent-enabled host in the network and resume its execution in that host. The agents can act on behalf of a user and execute autonomously according to their internal goals. Today the sophistication of mobile systems has increased tremendously over time as well as their associated security threats. Agent technology shares common ground with AN, especially, in the domain of security as we will demonstrate later in this paper. We are confident that a lot of security problems existing in the AN domain can be tackled with the use of research results accomplished within the agent domain.

Currently, research efforts to secure active networks such as Smart Packets [30], CONVERSANT [31], SANE [32], Ensemble [33] and ANTS [46] depend mainly on the usage of cryptography to provide security services. We will demonstrate in this paper that cryptography is only one of many options that can be used alone or in combination with others in order to provide integrated security solutions for the emerging infrastructure of AN and cover its needs.

2. Active network infrastructure

2.1. Key elements in the AN Infrastructure

Generally, we can distinguish three key elements in the AN infrastructure:

Execution Environment (*EE*). This is the place where the active code executes. The EE offers access to the core node resources via a policy-controlled scheme. This can be for instance a mobile agent system that takes care of the execution of an agent. Other EEs can coexist such as ANTS, ALIEN etc.

Active Code (*AC*). This is the code that is actually executed in the EE of the node. The code could be written in any general-purpose language e.g., Java, C, etc, as long as the EE supports it or even contain references to code already installed in the active node. By execution in the EE, the code programs the node according to user preferences.

Active Code Carrier (*ACC*). The active code is carried from the source host to the destination host. There are two ways of actually moving the code to the target node known as the in-band and out-band programming methods.

In the *in-band programming* method the active code is integrated into every packet of data sent to the AN node (also known as the capsule approach [49]). The EE on the node executes the program and adopts the functionality of the node for the specific packet or the specific flow. This approach is the most flexible one but these programs are very small due to the size limitation of the packets. That, in addition to the transport overhead, makes the programmability based on capsules limited, especially in connection-oriented communication environments like ATM where re-configuration/programming of the AN nodes is needed much less frequently than processing a packet payload.

On the contrary, in the *out-band programming* method the active code is injected in the AN node in a different session from the actual data packets that it affects. The user could install the desired code any time on the node. This code would then execute based on internal (e.g., according to node's EE schedule) or external (e.g., user activation command) events and program the node to process the desired data selectively. The data is recognized by specific tags or even by categorization, e.g., all data coming from a specific node. The agent approach which is discussed in this paper falls within the out-band programming category.

2.2. The nature of active code

Active code can have different characteristics. It can be:

Stateless. AC is a dumb program that is simply transferred from node to node where it is executed every time from the beginning.

Statefull. Here the AC maintains its state while it traverses the network and can make decisions based on that state. This gives a dynamic nature to the AC as it can stop its execution in one node and continue it in another with respect to environmental conditions.

Stationary. The AC permanently resides within one node, takes requests via specified interfaces and then programs the active node.

Mobile. The AC can freely move around from node to node at its own will and according to its internal goals.

By combining the statefull and the mobile attributes, our AC obtains the characteristics of a mobile agent. Mobile agents can move around and execute tasks autonomously and according to their goals which can be dynamically changed. Furthermore, mobile agents can also be intelligent, and that makes them even more interesting for AN. An intelligent piece of code that moves around as AC can be considered as the most advanced form of AC. All other forms derive from this one by combining some but not all characteristics mentioned within the intelligent and mobile agent research domain [26].

2.3. The role of agents in active networks

It is important to clarify that an agent can be an AC or not. Agents can slip easily in the role of AC, but also serve as middleware technology. No matter which one of the approaches described below we take, almost the same security requirements have to be fulfilled. Agents can be used in two ways in AN:

As active code carriers. Agents could be the vehicle that transports the code to be executed in the active node and modify its behavior. One could wonder, why on earth do so? The answer lies on the nature of agents. They have several characteristics such as fault tolerance, intelligence, etc, which could be utilized in order to achieve an aim. It is not very difficult to imagine the following scenario: an agent is dispatched with mission to carry code to be placed into a list of active nodes so that a network-wide service can be realized. Furthermore the agent should set the security context within which this active code can be modified/used/accessed in this node and return back to its owner with a result report. Doing this operation with the RPC-paradigm would require network availability as well as node availability. On the contrary, with agent technology network, availability is not a requirement anymore since the agent can execute autonomously, control its environment and act in the right time according to its goals. Furthermore, if data changes while the agent is in transition (e.g., a node on agent's list requests that another node which is not on the list should also have the specified code), it is easier for the agent to adapt its behavior without human intervention. Agents as carriers is a solution for heterogeneous environments where agents and code to be executed on active nodes are written in different languages.

As active code. Here agents are not the quasi-dump carriers, but they are the actual actors. They control directly node operation and configuration. The agent itself is the active code that executes in the EE and programs the active node.

2.4. A multi-EE active network architecture

The architecture of a multi-EE active node is presented in Fig. 1. Please note that one of the EEs is the agent EE in which AC is realized as an agent. The main components of such an architecture constitute of:

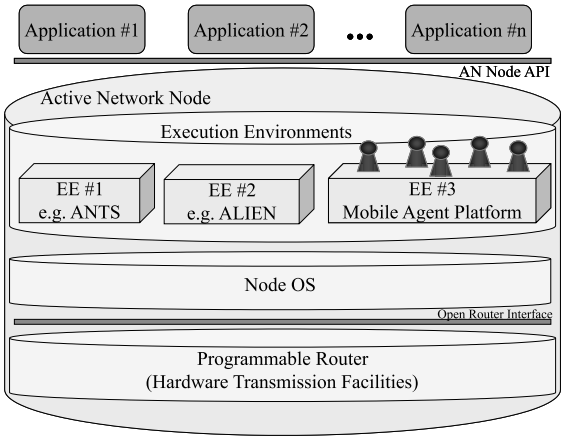

Fig. 1. Multi-EE active node architecture.

A programmable router. The router is accessed via an API for dynamic programming of it resources. The open node interface represents the abstraction of the router resources ranging from computational resources (CPU, memory, etc.) to packet forwarding resources (bandwidth, buffer, etc).

The NodeOS. This is the operating system running on each node (router) in an AN. The NodeOS provides the basic functionality from which the EEs built the abstractions presented to the active applications. The architecture of the NodeOS and its functionality is outlined in detail by the AN Node Operating System (Node OS) Working Group [34]. Let us mention that the NodeOS could also be a non AN specific OS like Unix or Windows NT, etc.

Execution environments are on top of the NodeOS, making use of its services. As noted [35] the functionality of the active network node is divided among the Node OS, the EEs and the active applications. The architecture allows multiple EEs of various providers to co-exist and be present on a single active node. Each EE (e.g., ANTS [46], ALIEN [47], Agent EE) exports a programming interface or virtual machine that can be programmed or controlled by third party code. The NodeOS manages the resources of the node. One of the EEs is the mobile agent EE where agents execute when they visit the node. The applications are able to access all the services offered by the EEs. Usually an application is bounded to one EE but we can foresee applications that will take advantage of the various characteristics of more than one EE and possibly combine their services.

Active code. Modules that execute in the EEs (agents in the case of agent EE) and via the facilities offered to them program the active node.

3. Threats in an active network infrastructure

Active networking supplies the users with the ability to download and execute code within a node. That is, by its nature, a security-critical activity. In such an infrastructure, the security implications are far more complex than in current static environments. In AN, the author of the active code, the user who deploys it, the owner of the node hardware, the owner of the execution platform (or even the execution place) can be different entities governed by different security policies and possibly competitive interests. In such a heterogeneous environment security becomes an extremely sensitive issue.

AC is transferred in some way to the node or is itself mobile, e.g., in the form of a mobile agent. Therefore, the attacks that AC and also the EE are susceptible to are more than those in current passive networks.

In general we can have:

- Misuse of execution environment by the active code,
- Misuse of active code by other active code,
- Misuse of active code by the execution environment,
- Misuse of active code and/or execution environment by the underlying network infrastructure.

Finally, a combination of the above categories is possible. This kind of attack (the complex and collaborative ones) is very difficult to detect, not to mention prevent or effectively tackle. Classical examples include the co-operation of various hosts and ACs against another EE or AC.

3.1. Misuse of execution environment by the active code

Malicious AC while executing in an EE can exploit security weaknesses in the host. It can perform attacks such as:

Masquerading. An AC may claim the identity of a trusted AC and therefore, be granted access to resources that is not entitled to. Such AC besides being objected to false security schemes can also badly damage the reputation of a legitimate AC in the community.

Denial of service. Malicious ACs may overuse intentionally all resources and services provided by the EE and degrade system performance. As a result, the platform cannot satisfy legitimate requests from other ACs. Furthermore, if e.g., the security service is blocked, further security breakouts could be introduced.

Unauthorized access. An AC that manages to bypass the authentication stage can harm the EE.

With various tricks or false language implementations [36], an AC can bypass authorization and authentication stages and obtain access to private data.

Complex attacks. Here, more than one AC cooperate in order to attack a host. These are the most difficult attacks as they are strategically planned and can be event triggered. These collaborative kinds of attacks are very difficult to identify not to mention to deal with.

3.2. Misuse of AC by other AC

These sets of threats are also very critical because of the variety of victims. An AC can perform various attacks against another AC including:

Repudiation. An AC can deny participation in a transaction or a communication, although this has actually taken place. An EE cannot prevent such an action but can provide sufficient evidence to assist with the resolution of such cases.

Masquerading. In the case of an AC to AC communication, one of the parties may try to disguise its identity and deceive the other one. Masquerading harms both the victim AC and the AC whose identity is being assumed.

Denial of service. An AC can send spamming messages to another AC or false requests to keep it busy and degrade the CPU power, disk space or AC's response time.

Unauthorized access. An AC not properly authenticated and authorized can directly interfere with another AC and perform various attacks, e.g., change AC's internal state, access/change its data, trap an AC and modify its configuration parameters or internal goals, steal info, etc.

3.3. Misuse of active code by the execution environment

An EE or node has complete control over the execution of an AC. Therefore, it can perform attacks such as:

Masquerading. An EE can masquerade as another EE in order to deceive an AC and obtain its sensitive information. If the AC cannot reliably verify the EE and it trusts the false environment it is given, it will surely be an easy target.

Denial of service. A malicious EE may ignore AC requests or introduce unacceptable delays to services. Deleting an AC or suspending it for enough time so that the operations the AC wanted to perform are not valid any more or have no meaning is an example.

Eavesdropping. The EE can monitor external/internal communications of the AC including every instruction executed by the AC. Therefore, it has access to all unencrypted or public data the AC carries. So, it can for instance, invade its privacy by fully accessing AC's memory space and acquiring info like electronic money, secret keys, etc.

Data and state manipulation. A malicious EE can manipulate data and/or state of the AC and therefore, interfere with AC's normal execution or even worse control and guide AC's execution based on falsely given perspective of environment. Furthermore, the EE can interfere with AC's communications and alter them.

Cloning. Cloning an AC and then using the clone to analyze the original AC and its objectives is another type of attack.

We mentioned above the main threats that exist in an AN infrastructure. Of course, a combination of them makes it even more difficult to prevent or deal successfully with it. All above mentioned security breakouts are performed when an AC is interacting with an EE. The AC relies on the EE to transport its code safely and securely to the desired host or execute it correctly, and that places an amount of trust to the EE anyway.

3.4. Misuse of AC and/or EE by the underlying network infrastructure (external misuse)

Threats exist also while the AC traverses the network from host to host. One external attacker could perform all kind of attacks such as masquerade, denial of service, unauthorized access, copy and replay, alteration, etc. A not so superficial scenario is the following: EEs are run by a user, e.g., in a Unix host. By misconfiguration, the user that runs the EE allows other users to access and modify the files that are stored on disk, e.g., the policy files. Then, another user (local or even remote via compromised WWW scripts) could easily

change the policy file and allow his AC to execute with full access rights. The difficulty with these kinds of attacks is that they cannot be dealt with at all, as they use resources not controlled directly by the specific product (in this case the EE). A product that runs in a Unix environment is vulnerable to all kind of attacks via the security holes of the Unix system. Such kinds of attacks cannot be predicted by the designer of the EE or the AC and are also out of the scope of this paper.

4. Security requirements of the active network infrastructure

AN aim at allowing third party entities such as users or applications to insert code to the network and customize its behavior for their specific needs. Opening up the networks is a straightforward approach to the evolution of current static nets. However, there is a drawback and that is called security. Security problems have to be dealt with successfully if the AN are ever to leave the research area and be successfully applied to the real world.

Unfortunately, security is not a simple testable property like a Boolean variable's true or false value. There are variable levels of security as it also obeys the golden rule "one size does not fit all". Security has many parameters to be considered and each one of them has the power to jeopardize the whole system if it is not correctly handled. Therefore, a variety of security requirements has to be addressed.

4.1. General security requirements of active networks

The AN infrastructures come with a double status; that of a legacy networks (e.g., data transportation) and that of a highly programmable network model adjustable on the fly to application-specific requirements. Thus, the spectrum of threats for such a new network model is extended. It includes not only the threat models of the legacy node and network systems, but also those of general purpose computing engines (e.g., safeness). We try here to recognize most of these problems in both. The basic requirements for a secure network are:

Privacy/confidentiality. Private data of the active code should remain private. Information is only disclosed to users authorized to access it. No third party should be able to acquire info, by monitoring or other kind of sniffing/hacking techniques, about this data without permission from the active code and via purpose-specific interfaces. This covers the cases when the code executes in an AN node and when it is transferred from node to node. Both communication and execution environment should satisfy these needs.

Integrity. The active code as well its transport within the network should be protected from unauthorized or accidental modifications. Information (data and code) should be modified only by users who have the right to do so and only in authorized ways. If such a goal is not feasible in all situations it should at least be possible to detect tampering after it has occurred and before the AC can cause any harm by executing maliciously.

Accountability and non-repudiation. Users are accountable for their security relevant actions. A particular case of accountability is non-repudiation where responsibility for an action cannot be de-coupled from the action itself, e.g., be denied or be modified. Every interaction with the system and its entities should be uniquely identified, authenticated and audited.

Availability. The use of the AN node and of its services should not be maliciously denied to authorized users. Furthermore, resource management, controlled concurrency, deadlock management, multi-access, detection and recovery from faulty states or endless loops should be tackled.

Authentication. In a heterogeneous networking environment such as that of the AN, we have to distinguish and securely authenticate the entities that want to inject code into our AN nodes. Successful authentication is the first basic step that we can use to make critical decisions. Authentication guarantees that the system entities are the ones they claim to be.

Access control and authorization. We have to specify what are the access rights of the code in our system. That means, we have to explicitly state what the code can do or what the code cannot do. Via authorization, we control every action that the

active code tries to execute and based on the policy we allow or deny it. The authorization mechanism relies on successful authentication as the basis to perform its goal.

Secure communication. Intra- and inter-EE and AC communication has to be secured. This covers both the transportation of AC from EE to EE as well as the exchange of messages in order to accomplish its goals.

Quality of service. This is not straightforward because it does not apply in general and is partially fulfilled by the "availability" requirement mentioned before. However, we consider it significant. Some applications require a certain amount of bandwidth or a certain amount of memory in order to function properly. Therefore, they have some minimal requirements from the underlying infrastructure, which should be met in order to function as designed. By compromising those minimal requirements, the application might behave in an unpredictable way and this will lead to further security violations. Guaranteeing some notion of QoS would limit the number and nature of many security violations and would add flexibility to easier handle the remaining ones.

4.2. Active network extended security requirements

AN provide new challenges in the security experts. Their extended security requirements are presented below:

Trusted identification of active node neighbors. It is necessary to be able to securely identify dynamically the neighbor nodes. This will help to not only prevent spoofing attacks but also provide proof and accountability in cases of malicious actions. This calls for digital certificates, which the nodes can present in order to authenticate themselves.

Verification of the EE. The AC should have the ability to verify the environment where it executes. EEs should also be able to perform such a verification in case we want to set-up a virtual private network of EEs [37]. The verification process could take place before the AC is installed in the node (using trusted services in another node) or even during runtime. The AC may contain private data or may even be environment sensitive and release info based on its internal list of goals. Therefore, the verification of the EE is necessary for building secure infrastructures.

Secure transition/distribution of code from node to node. We have to make sure that the active code is transmitted securely from one node to the other. Also facts like integrity, alterations, etc. have to be taken in account. Today industry's de facto standard protocols like SSL [41] or TLS [42] can be used for this kind of operations.

Policy-based active code installation/de-installation. After the arrival of the active code carrier in the node, it should be checked whether it has access to install the AC to the node. The active code could be run immediately or installed in a local database for future usage. It should be guaranteed that only trusted users install or remove software on the node.

Policy-based active code invocation. Once the AC is installed in the node database, we need a policy scheme to say who can access this piece of code and under which policy conditions. Preinstalled pieces of code in the node could also be seen as extended libraries or services. Other ACs may require results from these components in order to achieve the goals. In that case, we have to specify which code has the rights to execute preinstalled node components and make use of their feedback. The policy could be set by the node or EE administrator, or even by the user who originally installed the code there. By being able to invoke other components we have more lightweight active code (since parts of services can be found dynamically at runtime and not be implemented in one big program) and we promote security (the pre-installed code could be set there by the node administrator who has tested it thoroughly).

Active code revocation. We should be able to maintain locally and network wide a list with revoked ACs. If a piece of AC behaves maliciously then the node administrator would forbid it to execute although its credentials (the user that signed it) may be valid. This could be done for various other reasons, e.g., we forbid execution of ACs coming from competing companies. These black lists could be pulled from/pushed in a central point within the network for network-wide usage.

Policy-based access to the node's resources. The managed resources in an AN node include: processor cycles, OS resources, memory, disk storage, input and output bandwidth, cryptographic hardware, other services and APIs which could be used via a dynamic policy based scheme. By controlling resource usage, we might be able to tackle partially problems like denial of service attacks.

Runtime access control for active code execution. When an AC executes we have to authorize every call it makes to system resources and services. That gives us the ability to provide flexible policies and different levels of access based on the author or the user of the code. Anonymous code will run in a sandbox with limited privileges, contrary to verified code, which can have special access to resources. Furthermore, the whole procedure should be dynamic in order to provide better response of the network to environmental conditions.

Prevent unauthorized interactions between EEs. Multiple EEs of the same or different type can coexist within the same node. We should make sure that these EEs do not affect each other and that the code executed in each of them does not result in unauthorized interactions with the code executed in another EE. This is the sandbox idea but this time between EEs. Each EE should have its own resources and manage them according to its needs. Preventing unauthorized interactions between various ACs that execute within an EE is EE's responsibility.

Network-wide management of security. There should be a way to enforce a policy or apply on the fly policy changes on the whole network automatically and with minimal human interference. This calls for mechanisms that support a distributed way of propagating policy or even support for a centralized policy. We have here two extremes. In the distributed scheme, each node has its own policy while in the centralized scheme each node pulls the policy from a central server. Of course there are other possible schemes that take advantage of both ways, e.g., co-existence of policies that could be pulled from or pushed to a third trusted node (not the central server). In that scheme, one could have policy references from node to node (a WWW connection style of policies). This eases the segmentation of the network to policy domains and simplifies the creation of virtual private networks with common policy schemes. In any case, this should be transparent to the network administrator.

Secure auditing. We should audit events occurring in the base of active node services as well as the events occurring because of actions taken by the AC. Decomposing auditing activity in this way, allows the active node base code to be simpler as it does not have to implement complex handling of audit messages. Audit logs should be securely stored possibly in a distributed scheme [38] (better survivability to attacks) and access to them should be policy based. Apart from the node audit, the active code may perform its own auditing and possibly report it via an interface to the node's audit facilities.

Safe code execution. This is a difficult goal to achieve. We have to make sure that the code that executes in a node executes correctly. There could be code that comes from trusted users, but that does not execute safely and compromises intentionally or unintentionally node security. Mechanisms that guarantee safe AC execution have to be used.

Dynamic policy schemes. It is desirable to have a dynamic way of managing access to the node and the network resources. The access policy should not be only a static Boolean result (access denied or accepted) but should be able to vary, in time or in a network-specific way. For instance, use of bandwidth resources should be more expensive (or require a higher privilege) to invoke during heavy load of the node. The economy based approach in which the AC has to pay for the resources it consumes might be just the right approach for the AN community.

Persistence. This service should exist within the node. During node shutdown the node should suspend all ACs and then, after rebooting, reactivate them and continue its normal operation. Otherwise, due to an accidental node reboot the AC could be lost forever. Persistence service provides a more reliable and fault tolerant active network infrastructure.

Predefined node manipulation. Many network operators are very much concerned with the idea of executing code within a node, mainly because of

the obvious or hidden drawbacks such an action carries. Thus, there is a need that specific interfaces are provided to the users via which they can interact with the node in predefined ways. The network operator itself installs the necessary code and services in the node and allows the user to call this code with predefined (and well tested) parameters. Although again we have code executing, we can predict the result of this execution since the node's status will change to one of the predefined ones. In order to make this idea more clear we can think of the following scenario: a user wants to install a compression filter for video transmission. Instead of providing his own implementation, the user calls the code already installed by the administrator with parameters that satisfy his goal, e.g., VideoCompression ⟨algorithm⟩ ⟨final_format⟩ ⟨time⟩.

Now the node administrator knows that the user has programmed the node in a specific and already known way. This can be seen as a hybrid approach since active code is executed (active network) but actually the node is manipulated via predefined interfaces (programmable network).

Hierarchically structured security services. Node security services should be designed and implemented in a hierarchical way so that they serve as basis for the development of more sophisticated services provided either by the node or by the applications. Such services include cryptographic services, resource access control, secure multiplexing services, etc., all in their simple form. Based on these services, an AN developer could use and combine them in a Lego-like way in order to offer new, optimized or customized APIs and services.

Support for anonymous principals. The existence of security should not be a drawback in the support of anonymity. We cannot expect everyone to sign the code he wants to execute in an AN node as the public key infrastructure (PKI) has not yet expanded as expected. Of course, since via policy we set access rights and this policy is mostly identity-based, an anonymous identity will have much less access rights than a normal user. Anonymous code will generally be considered as non-trusted code and be run within a sandbox with minimal access rights. Therefore, we position ourselves positively to the anonymous support and to selective control of its capabilities via policy.

Target specific code distribution. The code is encrypted and will be used only by the intended node or even better by the intended EE. A mechanism should exist that will allow the active code to select the EEs or nodes it executes in. If the AC falls into the hands of a non-intended party then, the code should be useless and not reveal private info.

5. Applying agent security to AN

Having presented the threat model, we will try here to see how we can deal with these problems. We will now present solutions that can be used in order to tackle the AC to EE, EE to AC and AC to AC attacks. Specifically, we will demonstrate existing methods of protecting (a) the EE and (b) the AC from the majority of all possible attacks. We will also try shortly to comment on these methods and their pros and cons.

5.1. Protecting the execution environment

By applying the agent technology to AN we now have an EE which is a place or an agency as defined in the MASIF [22] standard. The basic point is that ACs should not interfere with each other or with the EE. Therefore, both should be isolated or communicate only via well-known and defined interfaces. For this reason, the standard solution is a security manager. Via a security manager implementation, a number of conventional techniques can be realized. Such techniques include cryptographic methods to encrypt/decrypt info or to authenticate EE and AC, audit mechanisms to log security-relevant events and intelligent filters that can analyze that info and take decisions, mechanisms to control access to system resources and mechanisms that isolate processes from one another. Since AC can be a stationary or mobile agent, we can now apply new techniques that have appeared in last years.

Signed code. AC is signed at least with one digital signature. The authority that signs the code can be the author of the code, the user who

deployed it or even node administrators. With a digital signature, we can then verify that the code is from the person that it claims (authenticity) and that it has not been tampered with (integrity). A security manager uses, after successful authentication, the authority of the AC in order to make authorization decisions before runtime or, most usually dynamically at runtime. Access rights are granted usually based on per-identity basis or per group/role basis [28] (in that case the identity is member of that group). The security manager may require that more than one entities have signed the AC in order to grant access to resources. Of course, digital signatures require the existence of a PKI, since certificates containing the identity of an entity and its public key have to be issued, distributed, verified and revoked.

It is important to state that a signed AC only guarantees that it comes from the authority it claims, but this method cannot guarantee that the AC will execute without fault or errors. Unfortunately, for many users the signed code has gone beyond the stage of authenticity and is a form of trust in the software itself, which is rather dangerous.

Furthermore, only the static parts of the AC can be signed. The code can be assumed to be static, however, state and variables are dynamic and change during execution or per hop of AC.

State appraisal. If the AC is an agent, then, it carries its state also while traversing from node to node. The state appraisal [4] is a mechanism that allows the AC to decide what privileges it will need at a particular EE. The approach allows automatic detection of those manipulations that put an AC into unacceptable state. It relies on the fact that the state appraisal functions belong to the immutable part of an AC, whose integrity is protected. The EE uses these functions to verify the correct state of the incoming AC and to grant the necessary privileges.

Safe code interpretation. The idea here is, that dangerous commands can be made safe or denied for visiting AC. The EE interprets the AC and in this way, it has fine-grained control and can examine each instruction or statement and decide whether to execute it or not. Note that safety offered by this approach depends on the security policy implemented by the interpreter. JAVA is a type-safe language by design. However, various implementations of it have failed to enforce it properly [21]. Other safe interpreter systems include SafeTcl [18], AgentTcl [17], OCaml [19], PLAN [20] and SafetyNet [48]. The drawback of this approach is the performance overhead when contrasted with compiled machine code.

Fault isolation. This technique, also known as sandboxing [15], isolates the memory domains where the program executes. Access is allowed only to the specific memory addresses and to data and code segments within their distinct fault domain. In that way, even programs in non-trusted languages such as C can be executed safely. The earlier versions of JAVA followed this approach. It provides AC safety with higher performance than the interpretation technique discussed previously.

Proof carrying code (*PCC*). The PCC [27] is a very promising approach that obliges the AC author to prove that the program possesses safety properties stipulated by the code consumer. Instruction overhead of sandboxing and policy checking can now be avoided. However, the platform independence is sacrificed in order to gain performance, but the benefits seem to outweigh this disadvantage.

Path histories. This approach [16] uses an authenticatable log of the previously visited EE in order to help the newly visited EE in decisions such as, whether to execute the AC and what resource constraints to apply. Once more, this technique can detect tampering but the final decisions rely on the EE and whether it trusts the other EEs already visited by the AC.

Resource management. We want to protect the resources of the EE from unauthorized usage or even set some quality of service constraints. There are methods [29] that provide such selective and policy-based usage of resources. Another popular approach is the economy-oriented one where the AC has some electronic cash and it pays for the resources it uses. This approach guarantees that at some point the AC will stop executing (due to lack of cash) or use the resources wisely (since it has to pay per use). Other approaches [1,2] make direct modifications to the JAVA virtual machine (JVM)

in order to add customized resource management. In this way, we can avoid or limit denial of service attacks.

5.2. Protecting the active code

Protecting the AC in its agent form is a difficult task to achieve. Here, we cannot apply the traditional security mechanisms because it is simply not normal that the execution environment attacks the application (AC is the application within the AN domain). Within the agent research domain this kind of threat is more or less tackled and the results can be used also in the AN area. With respect to the autonomy and free roaming of the AC we have the following general purpose techniques:

Partial result encapsulation. Here, the results of the actions of the AC are encapsulated. By encapsulating the results, we cannot prevent the EE from misbehaving but at least we are able to detect tampering afterwards and take the appropriate actions. In order to minimize the amount of data to be encrypted sliding encryption [5] can be used. Another method is to use partial result authentication codes (PRAC [6]). In addition, the EE could be asked to encapsulate [7] the results while AC traverses the node. Furthermore, a platform-oriented technique based on improved PRAC has been developed [8].

Shared secrets and cooperation. Here, the idea is that the AC's itinerary is recorded and tracked by another cooperating AC and vice-versa [9] in order to support each other. An AC does not possess all the elements needed to fulfill the task alone. By dividing a task between two ACs there is better chance to detect malicious behavior of EE. One of the ACs has to execute in a trusted EE and securely communicate with the other one. The scheme can be easily generalized to more than two cooperating ACs. In another approach [10], in order to perform a task we use multiple replicas of that AC. Although many replicas might be destroyed by malicious EEs, enough replicas will remain to bring the desired task to its end. This technique ensures that at least one AC will reach its destination and possibly fulfill its goals. This approach shares similarities with the path histories method mentioned before, but extended with fault tolerant capabilities. Furthermore, it applies only to scenarios where AC can be duplicated without problems. Here, survivability of AC is the main aim (suitable military applications), even with the obvious drawbacks of resource over-consumption and task repetitiveness.

Execution tracing. The EE has a non-repudiatable log where operations of the AC are logged and signed by the EE. With this technique [11] we are able to detect unauthorized modifications of code and state. The traces are used in order to identify the malicious EE if suspicious results occur. This method, although designed for the AC, provides also some sort of protection to the EE if traces of the involved EEs can be obtained and analyzed.

Environmental key generation – clueless AC. This technique [12] concentrates in designing AC in such a way that upon detecting an environmental variable, a key is generated and is used to unlock some cryptographically protected executable code. A third party that reads the AC's code cannot uncover the triggering mechanism. One problem with this approach is that you can actually modify the EE (if you have access to the source code) and instead of executing the code after removing protection, you just print it or save it in a file. As said before, this assumes access to the source of the EE which is not the general case for commercial programs. Furthermore, this approach does not provide privacy nor guarantees the integrity of the code during execution, but is well suited for event-triggered actions based on environmental sensing.

Computing with encrypted functions/data. Here [13] the idea is that AC passes to the EE an enciphered function to execute. The EE cannot decipher the original function that AC wants to compute. This approach differentiates between the function and the program that implements that function. The goal is to encrypt functions in such a way that their transformation can again be implemented as programs. Similar is the approach of computing with encrypted data.

Obfuscated code. The idea behind this approach [14] relies on obfuscation which is a mechanism that transforms an application into one that is functionally identical but much more difficult to

understand and therefore to attack. After an AC protection interval, the AC sensitive info becomes invalid and is useless. The drawbacks of the approach is that (a) it can be used only for temporary data that can be invalidated, (b) it cannot use any external libraries (this would contradict obfuscation) and (c) a meaningful optimal interval has to be found (this is task specific).

AC self encryption. This approach is based on the idea of creating secure channels between EEs via traditional cryptograhic means. An AC can encrypt itself or critical parts of itself using destination EE's public key. Then the ciphertext is put in a clearcode wrapper that transports the AC to the destination EE. On arrival there, the AC decrypts and executes. This approach guarantees the secure transition to selected EEs (as only there the code can be decrypted) but requires that each EE (or node) has a pair of public/private keys.

Privacy sought via this approach can be accomplished also without encryption although all code bits are sent in clearform over normal networks. The idea in this approach [43] is that by "chaffing and winnowing" data streams the attacker cannot find out which bits belong to which stream and only the intended receiver is able to filter out the meaningful bits.

Dedicated hardware/smartcards. With the use of tamper-resistant and verifiable hardware [25] it is possible to protect the AC. For instance, a tamper resistant smartcard with a cryptographic co-processor can be attached to the node. Smartcards provide a trusted EE where security functions can be outsourced. AC can send security sensitive code fragments to the smartcard and execute there. It is obvious that smartcard manufacturers should be an independent entity in the whole process. Usage of smartcards is not panacea as they face their own security problems [23].

Anonymous/untraceable AC support. The AC can itself implement encryption functions and services. AC can implement Chaum's MIX [24] nodes that hide the correlation between incoming and outgoing AC. In this way, it is possible to deploy anonymous ACs that traverse the network and report back to their origin, without revealing their identity to the recipient. Anonymous ACs, of course, will probably be given minimal (if any) access rights from EE administrators.

6. Summary and conclusions

We have presented here an approach that tries to combine two domains, that of AN and that of agent technology. Both domains share common ground, especially, when it comes to the security issues. The AC that executes within the AN nodes can be seen in its most advanced form as an intelligent mobile agent. Therefore, we can apply most of the security techniques developed from the agent community and tackle successfully similar issues in ANs. As agents penetrate the AN infrastructure with a double status (as active code and as active code carriers) it is evident how complementary both approaches are.

We have analyzed the nature of AC and presented the key elements within a multi-EE AN node architecture. Then we tried to focus on the security threats that exist in such a heterogeneous environment in AC to AC, AC to EE and EE to AC relations. In all cases, most of common security attacks can be accomplished. Subsequently, we set the most obvious requirements of AN and tried to explore the agent-based solutions that effectively deal with the threats identified above.

We are convinced that by applying the agent technology to ANs many more advantages [39] can be obtained than just those presented here, relevant to the security sector. It is very likely that agent technology will play an important role in the development and expansion of active networks. The basic characteristics of agents such as mobility, autonomy and intelligence can push networks to become "open", active and more powerful. Because agent technology advances continuously and has made significant contributions in the area of code mobility and security, it would not be wise to ignore this fact and try to reinvent the wheel every time in every new approach we take. Seraphim [44] and BANG [40] partially integrate the agent approaches proposed here and aim at proving the advantages agents can bring to ANs. By integrating solutions already tested in other domains, we can build on top of these and provide

more sophisticated approaches, that tackle the ever increasing complex security attacks.

References

[1] P. Bernadat, L. Feeney, D. Lambright, F. Travostino, Java sandboxes meet service guarantees: secure partitioning of CPU and memory, Technical Report 980, The Open Group Research Institute 1998, http://www.opengroup.org/.

[2] D. Milojic, G. Agha, P. Bernadat, D. Chauhan, S. Guday, N. Jamali, D. Lambright, Case studies in security and resource management for mobile objects, The Open Group Research Institute, University of Illinois at Urbana-Champaign.

[3] Cetus links on mobile agents: http://www.cetus-links.org/oo_mobile_agents.html.

[4] W. Farmer, J. Guttman, V. Swarup, Security for mobile agents: authentication and state appraisal, in: Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS '96), September 1996, pp. 118–130.

[5] A. Young, M.Yung, Sliding encryption: a cryptographic tool for mobile agents, in: Proceedings of the Fourth International Workshop on Fast Software Encryption (FSE '97), January 1997.

[6] B.S. Yee, A sanctuary for mobile agents, Technical Report CS97-537, University of California in San Diego, April 1997.

[7] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, G. Tsudik, Itinerant agents for mobile computing, IEEE Personal Communications 2 (5) (1995) 34–49.

[8] G. Karjoth, N. Asokan, C. Gülcü, Protecting the computation results of free-roaming agents, in: Proceedings of the Second International Workshop on Mobile Agents, Stuttgart, Germany, September 1998.

[9] V. Roth, Secure recording of itineraries through cooperating agents, in: Proceedings of the ECOOP Workshop on Distributed Object Security and Fouth Workshop on Mobile Object Systems: Secure Internet Mobile Computations, INRIA, France, 1998, pp. 147–154.

[10] F.B. Schneider, Towards fault-tolerant and secure agentry, in: Proceedings of the 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, September 1997.

[11] G. Vigna, Protecting mobile agents through tracing, in: Proceedings of the Third ECOOP Workshop on Mobile Object Systems, Jyvälskylä, Finland, June 1997.

[12] J. Riordan, B. Schneier, Environmental key generation towards clueless agents, in: G. Vinga (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, vol. 1419, Springer, Berlin, 1998.

[13] T. Sander, C. Tschudin, Protecting mobile agents against malicious hosts, in: G. Vinga (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, vol. 1419, Springer, Berlin, 1998.

[14] F. Hohl, Time limited blackbox security: protecting mobile agents from malicious hosts, in: G. Vinga (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, vol. 1419, Springer, Berlin, 1998, pp. 92–113.

[15] R. Wahbe, S. Lucco, T. Anderson, Efficient software-based fault isolation, in: Proceedings of the 14th ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review, December 1993, pp. 203–216.

[16] J.J. Ordille, When agents roam, who can you trust? in: Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, OR, May 1996.

[17] R.S. Gray, Agent Tcl: a flexible and secure mobile agent system, in: Proceedings of the 1996 Tcl/Tk Workshop, July 1996, pp. 9–23.

[18] J.K. Ousterhout, J.Y. Levy, B.B. Welch, The safe-Tcl security model, Sun Microsystems, March 1997.

[19] The Caml language, http://pauillac.inria.fr/caml/index-eng.html.

[20] PLAN a Programming Language for Active Networks, http://www.cis.upenn.edu/~switchware/PLAN.

[21] V. Saraswat, Java is not type-safe, ATT Research, http://www.research.att.com/~vj/bug.html.

[22] MASIF – Mobile Agent System Interoperability Facility, http://www.omg.org/docs/orbos/98-03-09.pdf.

[23] B. Schneier, A. Shostack, Breaking up is hard to do: modeling security threats for smart cards, in: Proceedings of the First USENIX Symposium on Smart Cards, USENIX Press, http://www.counterpane.com/smart-card-threats.pdf.

[24] D. Kesdogan, J. Egner, R. Büschkes, Stop and go MIXes providing probabilistic anonymity in an open systems, in: Proceedings of the Second Workshop on Information Hiding (IHW '98).

[25] U.G. Wilhelm, A technical approach to privacy based on mobile agents protected by tamper-resistant hardware, Ph.D. Thesis No. 1961, École Polytechnique Fédèrale de Lausanne, 1999.

[26] H.S. Nwana, D.T. Ndumu, A brief introduction to software agent technology, in: N. Jennings, M. Wooldridge (Eds.), Agent Technology Foundations: Applications and Markets, Springer, Berlin, 1998.

[27] G. Necula, P. Lee, Safe kernel extensions without run-time checking, in: The Proceedings of the Second Symposium on Operating System Design and Implementation (OSDI '96), Seattle, WA, October 1996, pp. 229–243.

[28] E.A. Kendall, Role modeling for agent systems analysis, design, and implementation, in: Proceedings of the First International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agents, October 1999, pp. 204–218.

[29] A. Tripathi, N. Karnik, Protected resource access for mobile agent-based distributed computing, ICPP Workshop 1998.

[30] BBN Smart Packets, http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html.

[31] CONVERSANT Project, Open Group, http://www.opengroup.org/.
[32] D.S. Alexander, W.A. Arbaugh, A.D. Keromytis, J.M. Smith, Secure active network environment architecture: realization in switchware, IEEE Network Magazine (1998) (special issue on Active and Programmable Networks).
[33] M. Hayden,The ENSEMBLE system, Ph.D. Thesis, Cornell University, January 1998.
[34] L. Peterson (Ed.), Node OS Interface Specification, AN Node OS Working Group, January 24, 2000.
[35] K.L. Calvert (Ed.), Architectural Framework for Active Networks, Draft version 1.0, July 27, 1999.
[36] Java security flows, http://kimera.cs.washington.edu/flaws/.
[37] S. Karnouskos, I. Busse, S. Covaci, Place-Oriented Virtual Private Networks, HICSS-33, January 4-7, 2000, Maui, HI.
[38] B. Schneier, J. Kelsey, Cryptographic support for secure logs on untrusted machines, in: Proceedings of the Seventh USENIX Security Symposium, USENIX Press, 1998, pp. 53–62.
[39] S. Karnouskos, Agent-populated active networks, in: Proceedings of the Second International Conference on Advanced Communication Technology, February 2000, Muju, Korea.
[40] BANG – The Broadband Active Network Generation project, http://www.fokus.gmd.de/research/cc/glone/projects/bang/.
[41] SSL v3 specifications: http://home.netscape.com/eng/ssl3/.
[42] RFC 2246, The TLS Protocol Version 1.0, January 1999, http://www.ietf.org/rfc/rfc2246.txt.
[43] R.L. Rivest, Chaffing, winnowing: confidentiality without encryption, April 1998, http://theory.lcs.mit.edu/~rivest/chaffing.txt.
[44] Seraphim project: http://choices.cs.uiuc.edu/Security/seraphim/.
[45] Active networks at DARPA, http://www.darpa.mil/ito/research/anets/.
[46] D.J. Wetherall, J. Guttag, D.L. Tennenhouse, ANTS: a toolkit for building and dynamically deploying network protocols, IEEE OPENARCH '98, San Francisco CA, April 1998.
[47] D.S. Alexander, ALIEN: a generalized computing model of active networks, Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA, December 1998.
[48] The SafetyNet project: http://www.cogs.susx.ac.uk/projects/safetynet/.
[49] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden, A survey of active network research, IEEE Communications Magazine 35 (1) (1997) 80–86.