An active networks security architecture

Arso Savanović, Dušan Gabrijelčič, and Borka Jerman Blažič
Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
(arso|dusan|borka)@e5.ijs.si
AND Stamatis Karnouskos
Frauenhofer FOKUS, Kaiserin-Augusta-Alee 31, 10589 Berlin, Germany
karnouskos@fokus.fhg.de

Keywords: active networks, security architecture, FAIN

Received: March 4, 2002

Active networks allow user-controlled network programmability. A security framework has to assure that our infrastructure will behave as expected and will efficiently deal with malicious attacks, unathorized attempts to execute active code etc. We present here a security architecture that is designed within the FAIN project and aims at supporting multiple heterogeneous execution environments. We argue for the pros and cons as well as why we have selected the specific components and also take a look at their interworking in order to provide the security services to the execution environments our active network node hosts.

A Introduction

Basic AN principles have serious consequences have serious consequences for the operation of an Active Network (AN). The possibility of loading and executing active code in Active Network Nodes (ANNs) imposes considerable threats to expected operation of AN/ANN due to flaws in active code, malicious attacks by unauthorized users, and conflicted code execution. Thus, security in AN deals mainly with protecting system (AN infrastructure) from malicious (unauthorized) and erroneous use. The central objective of FAIN [7] security architecture is to guarantee robust/secure operation of AN infrastructure despite the unintentional or intentional misbehaving of users, i.e. their respective active code/active packets.

Fulfilling these objectives is fundamental for the usability of ANs. Clearly, if it is trivial for any user to intentionally degrade the performance of an AN or any single ANN, or to bring down the AN/ANN, then ANs are not really usable. Note the difference between degrading performance of (disabling) an ANN and that of an AN. It is possible that specific network service, i.e. the respective active code, is consistent with local security policy of an ANN; however, due to global, network wide behaviour of the protocol, it can degrade the performance of (part of) network or even completely disable it. Furthermore, if an unintentional error in the design of a new network service, its implementation (active code), or its configuration can degrade performance of an AN/ANN or disable an AN/ANN, then ANs are not really usable.

Finally, if a malicious or unintentional misbehaving of any user can severely degrade or even disable the network services perceived by other user(s) of an AN but with no affect on the ANN/AN, then again, ANs are not really usable. The threat model for active networks covers three broad classes of security issues: protecting AN infrastructure from users and active code, protecting users and active code from other active code, and protecting users and active code from AN infrastructure. However, the scope of initial FAIN security architecture is limited mainly to the first class, i.e. protecting AN infrastructure from users and active code.

B FAIN Active Nodes

The FAIN Reference Architecture consists mainly of AA, VE, EE and Node OS:

- **Active Applications/Services** (AA) are applications executed in Active Nodes. An AA is often referred to also as Active Code (AC).
- **Execution Environments** (EE) are environments where application code is executed. A privileged EE manages and controls the Active node and it provides the environment where network policies are executed. Multiple and different types of EE are envisaged in FAIN. EEs are classified into Virtual Environments (VEs), where services can be found and interact with each other. VEs are interconnected to form a truly virtual network.
- **NodeOS** is an operating system for active node and includes facilities for setting up and management of channels for inter-EE and AA-EE communications, manages the router resources and provides APIs for AA/EEs, isolates EEs from each other. Through its extensions the NodeOS offers:

- **Resource Control Facilities (RCF).** Through resource control resource partitioning is provided. VEs are guaranteed that consumption stays within the agreed contract during an admission control phase static or dynamic.
- **Security Facilities.** Main part about security is authentication and authorisation of using the resources and other objects of the node like interfaces and directories. Based on the policy profile of each VE security is enforced.
- **Application/Service code deployment facilities.** As flexibility is one of the requirements for programmable networks partly realised as service deployment either on the fly or static, the NodeOS must support it.
- **Demultiplexing facilities.** It filters, classify and divert active packets. Flows of packets arrive at the node and they should be delivered to the VE and consequently to the service inside the VE they are destined for.

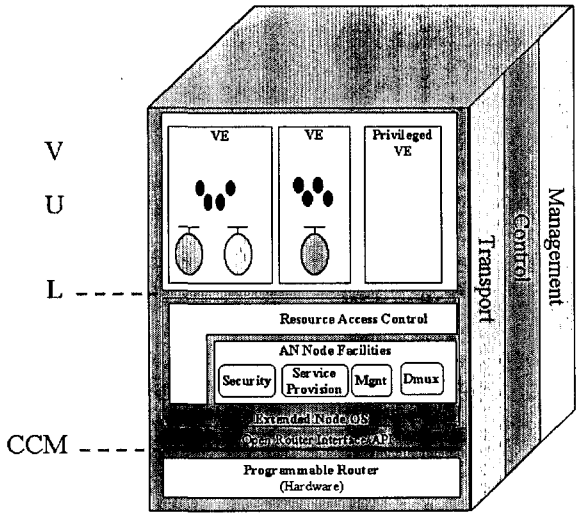

Figure 1: FAIN active node reference architecture

Figure 1 describes the main design features of the FAIN nodes. In FAIN a number of node prototypes are under development as follows:

- A high performance active node, with a target of 150 Mb/s.
- A range of flexible and very functional active nodes/servers, with the objective of supporting multiple VEs and hosting heterogeneous EEs.

The common part of the prototypes (the FAIN middleware) is the NodeOS with the relevant extensions.

C General FAIN Model and Security Requirements

The fundamental property of FAIN AN is the possibility to dynamically inject active code known as active application (AA), that implements new functionality, into the network. This code is executed or interpreted by specific execution environment within ANNs and this way it provides end- user applications with application specific network services. Many different active applications can coexist in an active network. We assume that FAIN AN consists of unlimited number of network nodes and some of them are active (ANN). Active code is injected into the network via active packets, which carry active code itself or its reference, which is used by ANNs to install the code from code repository. Code can be executed in the nodes within the packet path. Execution provides new functionality in the network, which can be temporary or permanent. It can also produce new packets. Each execution uses some of the ANN and AN resources, like CPU, storage and bandwidth, again temporarily or permanently. Specific code in an ANN can be injected, removed or replaced by explicit or implicit request. Additionally, the following properties apply to generalized AN model [1]:

- an AN is a distributed system
- an AN is a packet-switched network, as opposed to circuit-switched
- not all nodes in an AN need to be active
- an AN explicitly provides for computation inside the network, but
- the primary goal of active networks is communication, not computation
- the contents of an active packet can legally change inside ANNs[1]
- not all packets are active
- an AN consists of multiple domains, each controlled by a different administration.

Active networking supplies the users with the ability to install and execute program code within a network node. That by its nature is 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 can all be different entities governed by different security policies. In such a heterogeneous environment security becomes an extremely sensitive issue. The possibility of loading and executing active code in ANNs imposes considerable threats to expected operation of AN/ANN due to flaws

[1] In ANNs the payload (data part) can be changed also, not just header fields.

in active code, malicious attacks by unauthorized users, conflicted code execution etc. Thus, security in AN deals mainly with protecting system (AN infrastructure) from malicious (unauthorized) and erroneous use. The central objective of FAIN security architecture is to guarantee robust/secure operation of AN infrastructure despite the unintentional or intentional misbehaving of users, i.e. their respective active code/active packets.

Active Code (AC) is transferred 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 an active network node by the active code
- Misuse of active code by other active code.
- Misuse of active code by an active network node.
- Misuse of active code and/or execution environment by the underlying network infrastructure.
- Misuse of the Active Network as an entity.

Finally a combination of the above categories is possible. These kinds of attacks (the complex and collaborative ones) are very difficult to detect, let alone to prevent or effectively tackle. Classical examples include the co-operation of various hosts and ACs against another EE or AC. Threats can also be analysed from the perspective of a single ANN, and from the network- wide perspective. Of course, threats to a single ANN apply also to the whole AN (domain). However, network-wide threats can be more subtle and harder to combat, since they are based on the global, distributed nature of network protocols, and thus, their respective active codes.

In the initial phase of the FAIN project, only high priority security requirements have been addressed in detail:

- authentication
- authorization
- policy enforcement
- active code/packet integrity
- code verification
- audit.

We have compiled this list in light of the main objective of the FAIN security architecture, which is to provide secure and robust operation of FAIN AN infrastructure in spite of unintentional and malicious misbehaving of AN users, i.e. their respective codes. From this perspective, our criteria in assigning priorities can be summarized as follows:

- How subtle is particular security requirement, i.e. the respective threat "behind" the requirement?
- More subtle yields lower priority.

C.1 Authentication, Authorization, and Policy Enforcement

FAIN ANN is essentially a multi-user computing system. As in any such system, enforcement of access control is a requirement of high significance within every FAIN ANN. On the other hand, FAIN aims at developing a flexible system. In order to achieve the desired level of granularity we decompose access control in authentication, authorization and policy enforcement. These three security requirements have the highest-priority within FAIN security architecture.

C.2 Active code/packet integrity

Active code is executed within an ANN and performs actions on behalf of a user. Therefore, active code is the "carrier of activity" and as such, it is a powerful tool when misused by malicious users, which can potentially tamper with active code while it is in transit over the network. For instance, the whole access control system could be circumvented, if the original active code can be modified or swapped with any other code. Similarly, there are ways to obviate access control system by tampering with active packets, such as cut and paste attacks and replay attacks. This is why protecting integrity of active code and packets deserves a high- priority.

C.3 Code verification

Protecting the active code integrity is a first step to ensure non- modification of the transient code. However this is considered pretty basic and we need to go beyond that in order to achieve a high level of security. The active code has to be somehow marked and tightly coupled with one or more entities, based on which further security decisions can be made. The code carries credentials from these entities, which have to be verified in order to set the security context within which this active code can execute. As code verification is critical into taking further security decisions, this is considered a high-priority requirement for the FAIN security architecture.

C.4 Audit

The Audit Manager component is an integrated part of the security architecture. Via this component

- all events occurring from the usage of the security subsystem are implicitly logged for further future usage.
- It also provides an interface to explicitly log any other events coming from other parts of the FAIN architecture in a clear and homogeneous way.

Modern computer systems do not emphasize enough on the significance of the audit facilities. However audit tools help in realizing possible security leaks (or even preventing some) and make sure that mistakes are not repeated. We feel that within the AN community special care has to be

taken with audit activities and therefore it is also considered a high-priority security requirement.

D Technical aspects of active network security

D.1 Authentication

Authentication is a process of verifying an identity claimed by or for a system entity. Symmetric or asymmetric cryptography can be used for authentication. Symmetric cryptography is suitable only for closed systems due to its scalability problems. Thus, we use asymmetric cryptography in FAIN. This requires every AN user to have a public/private key pair and a valid public key certificate. Nevertheless, common remote authentication protocols employ a handshake, i.e. a two way communication in order to perform authentication. In active networks this would require an end-host to perform an authenticating handshake protocol with every ANN en route, which is clearly unacceptable. Thus, we propose the use of "unidirectional" procedure, where authentication is based on digital signatures and one-way communication from end-host to an ANN. Overview of this authentication scenario:

- User employs its private key to digitally sign the static part of an active packet and adds a signature to the packet it transmits
- ANN uses the public key certificate to verify the validity of the user's public key.
- If valid, ANN employs user's public key to verify the digital signature of the packet

A PKI infrastructure is needed to support authentication based on digital signatures.

D.2 Authorization

There will be several enforcement engines in FAIN ANN, each of them residing in a different FAIN ANN subsystem and responsible for mediating access to functions and resources of the respective subsystem. On the other hand, authorization component can be either integrated with policy enforcement or separated from it. In the former case, there would also have to be one authorization engine per ANN subsystem. In the latter case, only one, general-purpose authorization engine can be implemented and used by all policy enforcement engines.

We have adopted the latter approach for FAIN due to the following reasons:

- no duplication of work; this is especially important if we consider that design and implementation of any security component is a difficult and subtle task
- inherent flexibility as a consequence of separation of authorization from enforcement
- possibility of reuse of existing tools.

D.3 Policy enforcement

In the initial phase our discussion is limited to enforcement mechanisms up to and including FAIN Node facilities level, i.e. we currently omit the discussion of policy enforcement within EEs/VEs. Policy enforcement is the active component of security architecture that enforces authorization decisions and thus enforces the use of ANN resources, which is consistent with local security policies. We distinguish two types of resources, hardware and functional resources. Hardware resources include basic low-level ANN resources such as memory, storage capacity, CPU cycles and link bandwidth. Functional resources are high-level resources in the sense that they consume some portion of hardware resources. However, with functional resources it is not important how much memory or storage space they consume but rather what purpose they serve within an ANN, i.e. what function they provide. Examples of functional resources include:

- special purpose files, such as configuration files,
- policy entries in the policy database,
- ANN state,
- ANN API functions themselves, etc.

We note that all resources in an ANN, hardware and functional, are accessible at certain node interface. In order to prevent unauthorized use of ANN resources, policy enforcement has to be scattered across different ANN subsystems that provide specific subsets of ANN API functions. Thus, basic technical approach to policy enforcement is to add an "adaptation" software layer on top of every subsystem API, which then mediates access to node API functions. Whenever an ANN function is called by an "external" entity (such as VE, EE, active code), this software layer:

- intercepts the request (call to node function) and suspends it
- provides call parameters to authorization engine effectively asking for authorization decision; parameters include caller ID, called function name, object(s) name, amount of requested hardware resources, etc.
- when authorization decision is returned
  - if request is authorized, enforcement layer resumes the execution of the request
  - if request is not authorized, enforcement layer discards the request and thus prevents unauthorized actions from taking place

In addition to this "high-level" operation, policy enforcement also has to operate at low-level in order to enforce proper usage of low-level hardware resources. At the "lower level" enforcement is embodied in a more complex policing algorithm(s), which can control the scheduler(s) for specific resource and thus impose limits on resource usage by an entity.

D.4 Active packet/code integrity

In general, protecting integrity of active packet/code while in transit over network involves cryptographic operations. The most common approach is as follows:

- at the sending end—generate integrity protection token (data):
  - calculate a hash of the packet/code
  - encrypt the hash to protect it from modifications
  - send the encrypted hash together with the active packet/code
- at the receiving end—verify the integrity of the packet/code:
  - decrypt the hash that accompanies the received active packet/code
  - calculate a hash of the active packet/code,
  - compare the two hashes; if they differ active packet/code has been modified and should not be processed or allowed execution.

The hash value, which is carried along with active packet/code and is used for integrity, can be protected either by applying asymmetric encryption or symmetric encryption.

If asymmetric encryption is used, integrity protection is provided by digital signatures and there is no need for ANNs to maintain a private/public key pair.[2] ANNs only need to be able to obtain the certificate chain, which verifies the validity of the public key of the party signing the active packet/code. Thus, the advantage of asymmetric encryption is that it eases management of encryption and decryption keys. However, the downside is that asymmetric encryption is on the order of two magnitudes slower than symmetric encryption.

In case symmetric encryption is used, the encrypted hash is known as a MAC[3] value. However, this requires each ANN to maintain a non-compromised private/public key pair and a public key certificate. ANN uses asymmetric encryption to establish a shared secret key with the sending end. Thus, asymmetric encryption in this case is still used, but this time only to set- up a secret key for symmetric encryption. Additional downside of symmetric encryption is that integrity protection requires a negotiation phase before active packet/code can be injected into the AN.

[2]Note that other security requirements may/will impose this.

[3]Message Authentication Code.

In FAIN we have used a combination of asymmetric and symmetric encryption for active packet/code integrity, in order to leverage the advantages of both. The proposed approach is as follows:

- each ANN has a public/private key pair and a public key certificate
- each ANN maintains a shared secret key with every of its direct neighbouring ANNs; neighbouring ANNs employ asymmetric cryptography for establishing and updating shared keys
- the sending end signs active packet/code (using asymmetric encryption) and injects it into the AN
- the ingress ANN fetches the public key of the signer and verifies it against its certificate
- the ingress ANN then uses this key to check integrity of the received active code
- if active code is intact, ingress ANN calculates a MAC value, using a secret key it shares with the next hop ANN
- ingress ANN sends MAC value along with active packet/code and its signature
- every subsequent ANN
  - uses the secret key it shares with previous-hop ANN and checks integrity
  - calculates new MAC values using the secret key it shares with the next hop ANN
  - sends the new MAC value along with the active packet/code

This approach represents a trade-off between FAIN goals of security and performance. On one hand, the described approach is based on the assumption that trust exists between ANNs, which obviously reduces the level of security. However, this is a valid assumption at least in a single domain, which is under the control of a single authority. The trust within domain is applied by per-hop symmetric encryption. On the other hand, this approach is advantageous for ANN performance, since it leverages high speed of symmetric encryption algorithms. Furthermore, because (pre-established) per-hop shared keys are used, it effectively eliminates the symmetric key negotiation phase. Note that per ANN public/private keys and per-hop cryptographic calculations are used. However, since some parts of an active packet are dynamic, i.e. they can change at every hop, they cannot be protected with digital signatures and, thus, per hop integrity will have to be used, anyway.

D.5 Code verification

Verification can enable us to trust to some extent that the active code will behave safely and properly and that we can have some guarantees on its resource usage on the node and in the network. But we shall say in general that verification provides only enhanced trust in proper and safe code execution, which is usually not related to the trust in the user on behalf of which the code is executing. Code verification can help an ANN decide whether to run the newly received code. If the code fails the verification test, it is not trusted and it is dropped or alternatively it can run in an EE with minimal facilities available. In the latter case the EE is the same one that will be used to run anonymous active code. Broadly, code verification techniques can be classified into two groups:

1. Digitally signed code, so we trust the user, organization or repository that has signed the code. Digital signature can be checked at the NodeOS level, immediately after it is available.

2. Various other mechanisms that can enhance the trust in proper and safe execution. These mechanisms mainly operate within EEs, and include techniques like proof carrying code, JAVA bytecode verification, and restricted languages.

If there is resource consumption estimate available, simple resource check is also possible. Since the scope of initial FAIN security architecture is limited to the NodeOS level, we propose the use of first approach, which employs digital signatures for code certification.

D.6 Audit

The information gathered by the audit manager are stored into the audit database and via a policy controlled way are available for further use. Decomposition of 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 not only locally on the node but also in a distributed scheme as this offers better survivability to attacks against the node. 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.

E FAIN Security Architecture

Figure 2 depicts a FAIN active network node; all shaded components are part of security architecture. As depicted, FAIN security architecture roughly comprises three parts: security subsystem, other ANN security components, and external security support facilities. Note that the scope of initial FAIN security architecture does not include EE layer of FAIN ANN architecture.

E.1 Security subsystem

Most of security critical decisions are made by security subsystem, which is one of several subsystems within an ANN. The Security subsystem is also responsible for management of security critical data, such as encryption keys, credentials, and policies.

This subsystem is the core of FAIN security architecture and includes the following components:

1. **Crypto Engine:** performs the actual cryptographic operations, such as symmetric encryption/decryption, asymmetric encryption/decryption, and hashing. It implements various cryptographic algorithms, which are used by other components in the security subsystem.

2. **Security Environment (SE):** in a secure fashion stores various encryption keys, which are required by crypto engine. For example, SE stores ANN's public key pair (private and public key) and all secret keys that an ANN shares with its neighbours (one per neighbour).

3. **SE Manager:** is used for managing the keys in SE. SE manager can provide facilities for manual configuration of encryption keys and can also automatically manage keys, e.g. by triggering a key exchange protocol with neighbouring ANN.

4. **Integrity Engine:** checks the integrity of active packets and active code. It depends on integrity protection data contained within an active packet and on crypto engine to do the necessary cryptographic operations.

5. **Verification Engine:** performs code verification (at NodeOS level), if any. It may depend on special data contained within an active packet and on crypto engine to do the necessary cryptographic operations.

6. **Authentication Engine:** verifies the authenticity of active packets. It depends on authentication data contained within an active packet and on crypto engine to do the necessary cryptographic operations.

7. **Authorization Engine:** is responsible for making a decision whether a given user request to execute specific action or to access/manipulate particular object within an ANN is authorized or not. Authorization engine provides this "service" to all policy enforcement engines in an ANN.

8. **Policy database:** stores security policies, which govern who can do what in an ANN.

9. **Policy Manager:** when asked by the authorization engine, searches policy DB and returns all security policies, that are relevant for a particular request, which is currently subject to authorization. It also provides facilities for editing entries in policy DB, either manually by an authorized user, or automatically, i.e. download policies from a centralized policy server.

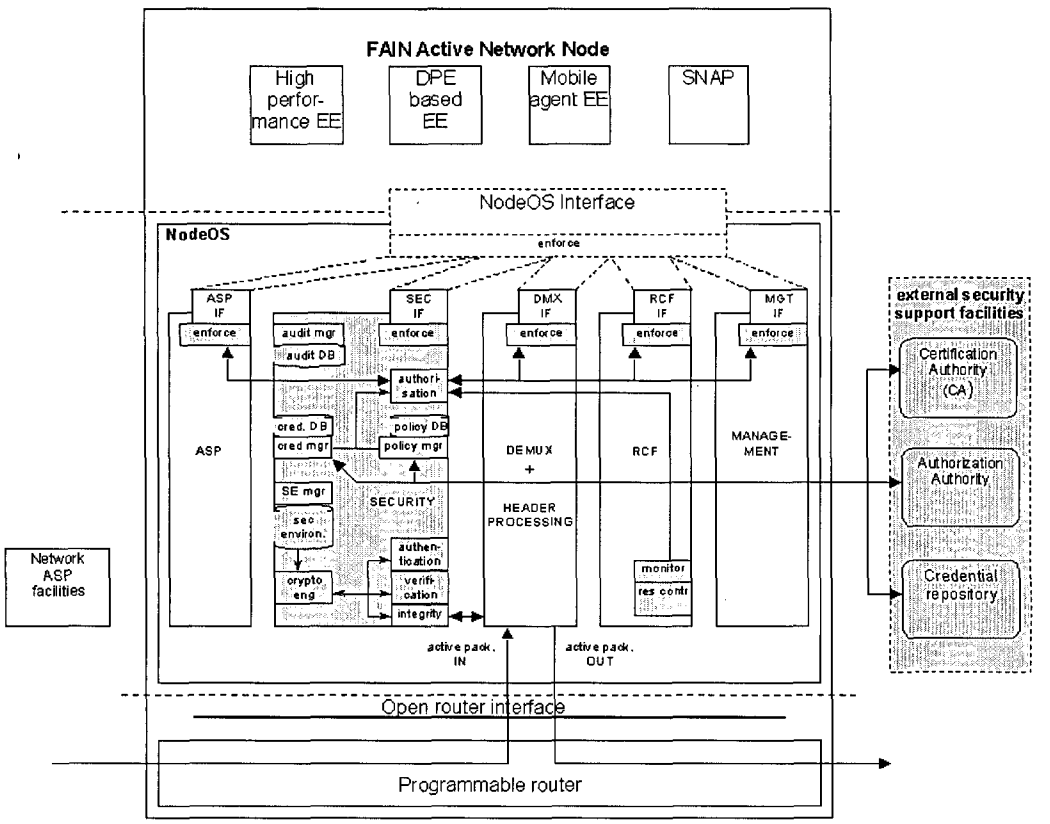

Figure 2: FAIN security architecture.

10. **Credential database:** stores users' credentials, such as public key certificates and attribute certificates.

11. **Credential Manager:** when asked by authorization engine, searches credential DB and returns all credentials, that are relevant for a particular request, which is currently subject to authorization. It also provides facilities for editing credential database, either manually by an authorized user, or automatically, i.e. search and download credentials from an external credential repository.

12. **Audit database:** stores an audit log of security critical events.

13. **Audit Manager:** will be the place where all security architecture's components audit their function in order to be used later in resolution of problems or even to make decisions. E.g. an Intrusion Detection System would use a view of the audit DB in order to recognize attacks against the system. The audit could be also distributed for survivability reasons.

E.2 Other ANN Security Components

The second part of security architecture includes components that are part of ANN but are external to security subsystem. This includes policy enforcement engines and various components providing environment variables, e.g. resource usage monitor. Various subsystems within an ANN offer their services and objects for use by users via their interfaces. Access to these objects and services is governed by security policies. Thus, enforcement of node security policies has to be performed at the point where they can be violated, i.e. at interfaces. At every ANN subsystem interface, a policy enforcement engine acts as an adaptation layer, which is responsible for mediating access to subsystem services and objects based on the authorization decision. While authorization is only a decision making, enforcement is an active process that prevents access to services and objects by unauthorized users. The Enforcement engine suspends the request at interface, asks the authorization engine whether this request is allowed and acts upon authorization decision, i.e. either allows or denies execution of the request.

In addition to these "high-level" enforcement engines, there are also "low- level" enforcement engines, which are tightly coupled with specific hardware resources available within an ANN and therefore they are considered as part of Resource Control Framework (RCF). Finally, there are some components in an ANN, which provide authorization engine with necessary data to make authorization decision. For example, resource usage monitor provides data on current hardware resource consumption by particular user, and a clock provides current time and date.

E.3 External Security Support Facilities

In the initial security architecture, we envisage these security support facilities:

- Certification Authority (CA)
- Authorization Authority
- Credential repository

Authentication based on digital signatures requires a user to have a public key pair and a valid public key certificate. Public key certificate binds a public key and an identity of its owner; these certificates are issued by a trusted third party called Certification Authority (CA). When a user enters an ANN, he must present his public key certificate to authentication engine. Alternatively, he can provide a pointer to his public key certificate in the form of a reference to certificate repository.

In the initial phase of FAIN, a single CA is sufficient for demonstration and testing purposes. This can be later extended with more CAs forming a fully-fledged Public Key Infrastructure (PKI).

Similarly, a scalable approach to authorization requires a user to have one or more attribute certificates. Attribute certificates bind public keys directly to privileges, which can be exercised by the owner of the key. Attribute Certificates are issued by a trusted party called Authorization Authority, which may not necessarily be the same as the Certification Authority. When a user enters an ANN, he must present one or more attribute certificates either directly or by reference to a repository. Later, when a user tries to execute an action, attribute certificates are used by the authorization engine to decide whether he has the necessary privileges.

The Credential repository can store both, public key certificates and attribute certificates. Repository can be implemented in many ways, such as a directory service or a web repository.

F Operation of security architecture

Basically, there are two checkpoints where security functionality from figure 2 is employed to protect an ANN: when a user enters an ANN and when a user tries to execute some action within an ANN. The former is represented by an arrival of an active packet in ANN and we call it *entry-level protection.* The latter occurs when a request for certain operation arrives at NodeOS interface and we call it *execution-level protection.*

In addition to these two types of security protections, one can distinguish two operations, which do not directly provide any security protections. Rather, these two are a sort of "backplane" operations, which support entry- and execution-level security protections. These support operations are: setup of a shared secret key between neighbouring ANNs and obtaining the missing credentials from a node external repository. A secret key, which is shared by a pair of neighbour ANNs, is used for hop-by-hop symmetric encryption of portions of active packet, which is leveraged e.g. for integrity protection. To setup a shared secret key between two ANNs, any key exchange protocol can be used. Key exchange has to be performed when a new ANN is added to/removed from the AN and whenever the key lifetime expires.

On some occasions, a situation may arise, when the credentials needed to make authorization decision are not present in an ANN. In this case, the missing credentials have to be searched for and obtained from somewhere in the network, usually from a repository service.

Finally, there is an audit facility within FAIN ANN, which is responsible for keeping a log of all security critical events within an ANN. This information is required for activities such as intrusion detection and analysis and assessment of security breaches.

F.1 Entry-level security protections

Figure 3 depicts a sequence of security operations that are performed for every packet that arrives at ANN. These security checks are aimed at detecting anything suspicious about this particular packet and, if so, discarding it. A packet is only delivered to appropriate EE if it passes all checks. Upon entering an ANN, an active packet is first processed in order to extract information needed for security checks. This information includes:

- Digital signatures, which are used for authentication, integrity, and verification
- MAC values, which are used for integrity protection
- Public key certificate(s), which are used for checking digital signatures
- Attribute certificates, which are used for authorization

After this information has been provided to security subsystem, entry-level security checks are triggered. The security subsystem verifies credentials, checks integrity of active packet and active code, performs code verification (if any), and performs authentication and returns the result of these operations to the de-multiplexing subsystem. Only if all these checks are successful, the packet is allowed to "enter" an ANN, i.e. it is first processed at NodeOS level (e.g. IP processing) and then forwarded to appropriate EE for further processing. If any of security checks fails, this is reported to de-multiplexing system, which discards the packet.

F.1.1 Active packet integrity

Integrity protection is based on cryptography. Every packet carries along at least one special token, which is used for integrity protection. This token is in the form of a digital signature and/or a MAC (Message Authentication Code). In

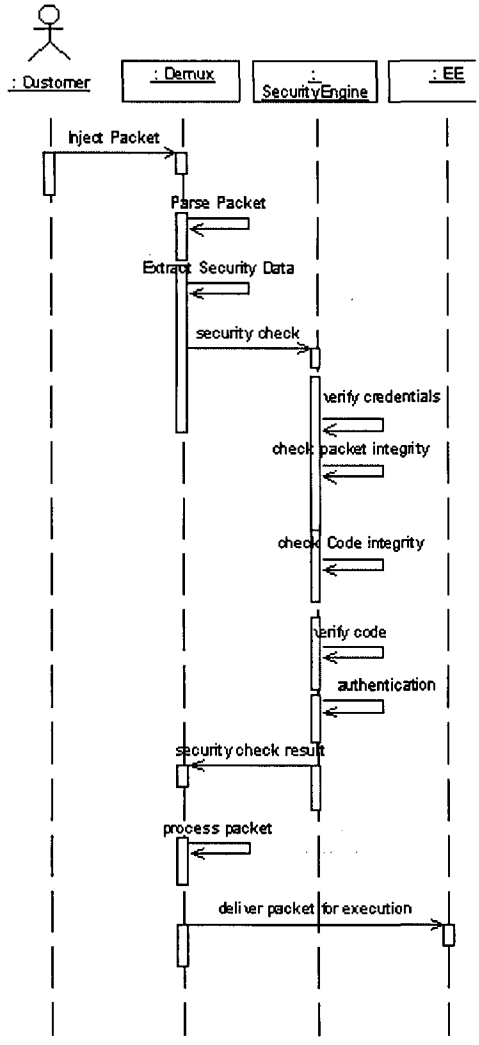

Figure 3: Entry-level Security Checks

FAIN bothe techniques are used for packet integrity. This is due to the fact that digital signatures are required for authentication, so it is sensible to leverage digital signatures for integrity as well. However, this applies only to static parts of an active packet, which do not change en route and can be signed by the source of the packet. For the dynamic parts of an active packet, which can change within an ANN, per hop integrity protections based on MAC must be used.

Integrity engine checks integrity in three steps. Firstly, it asks crypto engine, which performs all cryptographic calculations, to decrypt the integrity token, in this case a MAC value; crypto engine needs to get appropriate decryption key from ANN's security environment. This decryption process returns a hash of the packet as it was seen by its sender. Secondly, it asks crypto engine to calculate hash of the packet. The last step is to compare this hash against the decrypted token. If they are equal, then integrity of the packet can be assumed. If these two values differ, however, then integrity check has failed.

F.1.2 Active code integrity and code verification

Here we check the integrity of the newly received active code. Note that integrity checks for active packet and active code need to be separate because of the fact that these protections are in most cases provided by different encryption keys, i.e. different actors. The reason for this is that active code can be tampered with even before it is included in any active packet. Thus, digital signature generated by packet source at packet creation does not suffice for active code. Every active code is accompanied by at least one special token, which is used for integrity protection. This token has a form of a digital signature and/or a MAC (Message Authentication Code). It is envisaged that both approaches to integrity will be used in FAIN. Digital signatures by code provider/manufacturer can provide integrity protection until code is injected into the active network. From there per-hop MAC protection can be used, which is expected to yield performance gains. Additionally, the advantage of per-hop MAC protection is that it covers both packet and code at the same time. Note that we omit the discussion of multi-domain issues in the initial phase.

When active code integrity is provided with digital signature generated by code provider, the integrity engine must first process code provider's public key certificate in order to extract and validate providers public key. After extracting a valid public key integrity engine checks integrity in three steps, similar to active packets. First, it asks crypto engine to decrypt the integrity token, in this case a digital signature. This decryption process returns a hash of the code as it was seen by the code provider. Second, it asks crypto engine to calculate hash of the code. The last step is to compare this hash against the decrypted token. If they are equal, then integrity of the code can be assumed. If these two values differ, however, then integrity check has failed.

The majority of active code verification techniques are

specific to particular EE. Since we have limited our current scope to NodeOS only, we do not address these mechanisms. The only general verification mechanism, which can be placed in the NodeOS is based on digital signatures by trusted parties. For the initial FAIN security architecture we use code providers as these trusted parties. This effectively eliminates the need for distinct code verification process within the NodeOS, since code provider's digital signature is checked as part of code integrity check.

F.2 Execution-level Security Protections

Once an active packet has successfully passed entry-level checks, active code(s) can execute and perform operations within an ANN on behalf of some user. Obviously, some users will have more privileges than others, i.e. security policies define who can do what in an ANN. In order to protect an ANN it is necessary to prevent users from abusing their privileges and violating security policies.

According to the previous paragraph, execution-level protection includes two steps:

- Evaluating every execution request against node security policies, which is performed by authentication engine and
- Allowing or denying execution based on positive or negative authorization decision, respectively; policy enforcement engines are responsible for this.

F.2.1 Policy enforcement

Crucial to policy enforcement is the subsystem specific enforcement engine, which is implemented as an adaptation layer mediating requests at subsystem interface. Every request at subsystem interface is intercepted and suspended by this adaptation layer. Before execution a request has to be evaluated against local security policies. Enforcement engine does not itself evaluate whether the request is compliant with local security policies. Instead it invokes the authorization procedure within the security subsystem and feeds it with request information, such as: requested action, name of target object and requesting caller ID. Only after authorization returns, "request authorized" does an enforcement engine allow execution of the request by the underlying subsystem. Obviously, if authorization returns a negative answer, i.e. "request not authorized", then enforcement engine simply discards the suspended request. This way, it prevents execution of unauthorized requests and essentially enforces users to adhere to local security policies.

F.2.2 Authorization

Here we check everything that is required to authorize the request, i.e. to decide whether to grant it or not. In flexible access control systems, authorization is not integrated with enforcement. Instead it is separated logically and in implementation. In this way, a single authorization engine can be used by multiple policy enforcement engines.

Authorization decision is based on the following set of data:

- request information (action, object name, caller ID)
- local security policies, which govern the way in which particular object can be used
- credentials associated with particular caller ID
- current values of environment variables, such as time of day and amount of resources used by subject

Enforcement engine provides the request information when it asks for authorization decision. This information is used as an "index" by policy and credential managers for fetching appropriate policies and credentials, respectively. Environment variables are provided to authorization engine upon request by facilities, such as system clock and resource monitoring module within Resource Control Framework (RCF) subsystem. Finally, after gathering all the required information, authorization engine processes this data according to its internal rules, which return a simple result, either saying, "request authorized" or " request not authorized." This is returned to the calling policy enforcement engine, which then acts accordingly.

G Related work

FAIN aims to develop a heterogeneous ANN, allowing coexistence of various technologies that enable installation and execution of active code within an ANN. Consequently, FAIN security architecture is aimed at providing a more general solution which provides necessary protections for such an heterogeneous system. This is reflected by the fact that security architecture we have presented does not incorporate details of specific EEs that exist in the FAIN ANN. Its goal is to be as EE independent as possible and provide a common set of basic security services required by all AN enabling technologies.

Some research projects on active networks have already tried to tackle the issue of security, in various ways and at different levels of completeness [3,4,6,8]. In contrast to FAIN, all these security architectures are tied to the specifics of the respective model of active networks and, consequently, reflect the original design decisions and, more importantly, trade-offs chosen by developers of the respective model. Java Security Architecture [5] proved to be useful for AN security, but it has some drawbacks in this context [6]. There has also been some more general work on AN security [2], but this work is still in the early phase.

Some research projects on active networks have already tried to tackle the issue of security [3][4][5][6]. Contrary to FAIN, all these approaches are tied to specifics of particular model of programmability. When designing a more general

AN security architecture, which is the case in FAIN, these specifics can not be assumed. Java Security Architecture [7] proved to be useful for AN security, but again it is technology specific and it also has some drawbacks [5]. There has also been some more general work on AN security [8]. This work is still in the early phase.

H Conclusion

We have presented in this paper a security architecture for future IP active networks as it is done in the context of FAIN project. We try to tackle the high priority security requirements such as authentication, authorization, policy enforcement, active code and active packet integrity and verification and last but not least audit. We have analysed the main design decisions that we have taken and the reasons why we decided to follow them. Subsequently we have presented the components of a security architecture that will be used by multiple heterogeneous execution environments within the same active node. We also provide a look in the interworkings of the architecture and its decision-making logic. A prototype implementation of the presented active network security architecture is currently under development, which will be used for exploring the advantages and drawbacks of our approach.

Acknowledgement

This paper describes work undertaken and in progress in the context of the FAIN - IST 10561, a 3 year project during 2000-2002. The IST program is partially funded by the Commission of the European Union. The FAIN consortium consists of University College London—UK, Jozef Stefan Institute—Slovenia, NTUA—Greece, Universitat Politecnica de Catalunya—Spain, Deutsche Telekom Berkom—Germany, France Telecom/CNET—France, KPN—The Netherlands, Hitachi Europe Ltd.—UK, Hitachi, Ltd.—Japan, Siemens AG—Germany, ETH—Switzerland, Fraunhofer FOKUS—Germany, IKV++ GmbH—Germany, INTERGAsys—Spain, University of Pennsylvania—USA.

References

[1] Active Network Working Group. Architectural Framework for Active Networks, Jul 1999.

[2] Active Networks Security Working Group. Security Architecture for Active Nets, May 2001.

[3] D. Scott Alexander, Wiliam A. Arbough, Angelos D. Keromytis, and Jonathan M. Smith. A Secure Active Network Environment Architecture: Realisation in SwitchWare. *IEEE Network*, Special Issue: Active and Programmable Networks:37–45, May/Jun 1998.

[4] B. Branden, B. Lindell, and S. Bernson. A Proposed ABone Network Security Architecture. ABone-draft, Nov 1999.

[5] Li Gong. Java security architecture (JDK1.2). Technical report, Sun Microsystems, Oct 1998.

[6] Active Networks Working Group. SANTS Security Overview, May 2000.

[7] Future Active IP Networks (FAIN) Project. http://www.ist-fain.org.

[8] Stephen Schwab, Richard Yee, and Rishi Dandekar. AMP Security Overview. Technical report, NAI Labs, May 2000.