An architecture for autonomic security adaptation

Une architecture pour l'adaptation autonome de sécurité

Andreas Klenk, Heiko Niedermayer, Marcus Masekowsky, Georg Carle
Computer Networks and Internet, University of Tübingen, Germany

Abstract

Abstract - Communication is the grounding principle of nowadays complex applications where the functionalities of the overall system are much more powerful then the ones of the isolated components. The task of keeping the communication system operable is highly critical due to the configuration complexity and the need for manual administration. Autonomous configuration mechanisms offer a compelling solution for the communication problem. We present an architecture for the autonomous configuration of secure, layer independent, end-to-end connections in this paper. The Extensible Security Adaptation Framework (ESAF) separates the particularities of communication setups strictly from the communication usage by the applications. Applications are unaware of the utilized security mechanisms and the complex configuration thereof. Protocols and security primitives can be easily introduced into the system whereas others might be disabled due to vulnerabilities without the need to modify existing programs. Moreover the setup can adapt to changing environments dynamically during runtime.
Keywords - Security Adaptation, Autonomous Configuration, Virtualization.

Résumé - La communication est l'élément de base des applications complexes d'aujourd'hui, dans lesquels des fonctionnalités du système entier ont une puissance beaucoup plus grande que celle des composants isolés. A cause de la complexité de la configuration et la nécessité d'administration manuelle, la tâche de tenir le système de communication en fonction est hautement critique. Une solution impérative pour le problème de communication est offert par des méchanismes de configuration autonome. Dans cette publication, nous présentons une architecture pour la configuration autonome des connexions fin-fin sécurisées et indépendantes de la couche. L'Extensible Security Adaptation Framework (ESAF) sépare strictement les particularités des environnements de communication et l'usage par les applications. Les applications sont ignorantes des méchanismes de sécurité utilisés et leur configuration complexe. Des protocoles et des primitives de sécurité peuvent être facilement introduits dans le système, tandis que d'autres pourraient être désactivés à cause des vulnerabilités, sans la nécessité de modifier des programmes existants. En outre, l'installation est capable de s'adapter dynamiquement aux environnements modifiants pendant le temps d'execution.
Mots clés - Adaptation de sécurité, configuration autonome, virtualisation.

I Introduction

During the last decade the number of computers drastically increased as well as the demand for communication. This trend raises the issue how to handle and guarantee security in those systems, considering their vast complexity. Manual configuration approaches reach their limit of applicability as the number of computer and the possible network interconnections rapidly grow. Hence security must be as easy to enforce as connecting a computer to the network.

With the introduction of innovative network concepts new challenges arise as more devices will be part of the network and network components will change more frequently. Sometimes the devices must act completely autonomic with little to none further administration, for example, in ubiquitous scenarios where computers and other electronic helpers are employed once and must operate autonomously further on. In such settings manual configuration is not feasible anymore. Thus, traditional manual security management approaches reach their limit of applicability.

The driving force for change is now the user with her needs and her fascination for new and innovative products. The complexity and the unforeseeable number of possible interactions and communication interfaces requires a self-configuration capability for the security and communication functions of the devices. A solution to this problem is offered by the introduction of autonomic communication.

I.1 The non-autonomic use of current technologies

Current security technologies are either integrated into the system (like IPSec) or into the application (like SSL/TLS). An example for security configuration at system level is the management of security policies or associations for IPSec. Configuration at the system level usually reflects the highest demand for security and therefore uses the protocol and cryptographic algorithm which offers the best security guarantees. The choice of the "best" security protocol reflects only badly the true security requirements at a given time. During a session only certain data need strong protection. Another example for potential lower security requirements is communication in trusted environments, say inside the company network.

The configuration, especially of IPSec, is awkward. It is up to the administrator (and additionally to the user in case of VPN clients) to configure the installation of IPSec as well as to provide device-specific and appropriate security policies. IPSec stores security policies in a security policy database (SPD). These policies determine if and how to secure a particular packet. Such a static approach ignores the particular requirements of the different stakeholders and cannot adapt to volatile requirements.

In contrast to the system-wide configuration, security at application-level must be supported already during the development by such programs. Applications which deal with the configuration of security protocols on their own are therefore forced to support the particularities of the protocol interfaces. Such applications break if old protocols become unavailable even if new security protocols are introduced into the system as a replacement. At the time of the development unforeseen security configurations cannot be used. This is ignorant of many use-cases and cannot adapt to new developments and the introduction of innovative security mechanisms. If an application is closely coupled with a security policy the administrator can only influence the behavior through application specific configuration options and cannot enforce the security policies at a single point in the system.

I.2 Autonomic Configuration and Adaptation

The basic principle of our approach to autonomic configuration is the consistent separation of communication requirements from the configuration of existing protocols. Applications specify their requirements agnostic of the implementation details of the available protocols. Application, administration, and user provide the system with *high level policies*. Given such an initial basic rule set ESAF [2] provides full autonomy and adapts the security mechanisms accordingly. Thus the framework is easily extensible with new protocols without the need to rebuild the applications.

Security requirements are influenced by all communication partners; the minimum requirements are therefore determined during a negotiation process. The partners agree upon a protocol choice and the corresponding configuration matching the most demanding security requirements of each participant.

Autonomous components must adapt their behavior in many cases to their environment. The logic how to determine the context could be realized by the application. The application can then modify its high level policies during runtime to reflect different trust levels.

I.3 Structure of the document

The remaining sections of the paper are organized as follows. First we give a short overview about current research in the field of Quality of Security Service in Section II. In Section III we introduce ESAF with its architecture and components. We describe how requirements can be expressed with policies and the API for applications. Next we discuss the applicability of ESAF for a typical scenario in Section IV. Finally we summarize our approach in Section V.

II Related Work

Levine was the first to discuss the effects of Quality of Security Service[7] in depth and to describe how to adapt the system to defined requirements. Her system offers security choices to reflect the different behavior of the mechanisms. The user, the application and the system determine the adequate algorithm jointly. The decision is derived from security ranges and performance ranges specified in form of policies. The core argument behind her reasoning is: If a user decides on a minimum security level for an application, would she ever agree to more security if that increases her costs?
The GSS-API[13] was developed prior to the QOSS approach to support applications with generalized security services. The per-message protection mechanism is enhanced by an option to take a so called Quality of Protection(QoP) parameter to select a particular security option based on performance values and the protection requirements of single messages. However, the use of the option is limited, because the QoP parameter depends on the implementation of the underlying algorithm which breaks the encapsulation of the mechanisms.
Ganz introduced the security broker[9] architecture for WLANs. This framework chooses security services upon security requirements specified by the user, available network performance and the performance of security routines. There exist three security classes to choose from, each with a different level of protection and different performance. Another contribution to adaptive security in WLANs was published by Saxena in [5]. The required security level depends mainly on the trust in the environment, say how hostile one expects the environment to behave. The paper argues that the spare processing and transmission resources are wasted in mobile environments if security is over-provisioned. Hence the tradeoff between security and performance is essential for the choice of security services.

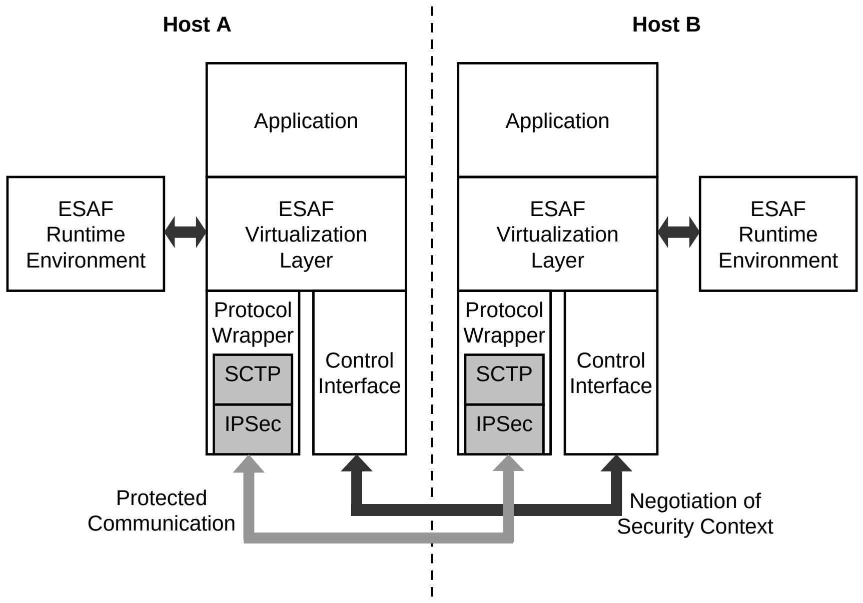

Fig. 1. Extensible Security Adaptation Framework/Extensible Security Adaptation Framework

III Extensible Security Adaptation Framework

The ***E**xtensible **S**ecurity **A**daptation **F**ramework (ESAF)* was designed to provide applications with a novel interface that provides virtualization especially for security. Applications can take control over the security protocols without the need to know anything about the parameters and interfaces of the protocol at all. The decision which protocol and which configuration should be used has to be derived directly form the security and communication requirements of the different stakeholders in the system: user, application, system, communication partners and many other instances determine the configuration needs.

These requirements are defined in *high level policies*. These policies describe in an abstract form the required security and communication parameters. The ESAF can map these *high level policies* internally onto *system capabilities policies* to derive the particular configuration that must be applied to the protocols.
Virtualization offers a compelling solution to solve two problems at once. The security and communication requirements must be formulated independently from a particular protocol, but they must still be expressive enough to state the requirements in necessary depth. The usage of the security protocol must be generic enough to replace the protocol in the system without the need to reconfigure or rebuilt the existing applications.
Exchangeability of the protocols only works if the necessary configuration does not introduce hard dependencies to a specific protocol. The socket interface achieves today a certain level of abstraction for applications but only after the socket is established. The initialization is done through a protocol specific sequence of commands including the *sockopt()* option. It is not possible in this case to exchange the old protocol with a new innovative and unforeseen protocol without breaking the application. This is especially true in cases when, for example, security was formerly provided through SSL, but now IPSec is the only secure communication option. Most often such an exchange is not possible at all.
Our solution is to introduce similar interface to the standard socket interface but offer generic configuration with *high level policies*.
ESAF not only acts system local, but can also assist the applications with the establishment of secure connections. A security context negotiation is performed during connection setup to determine the requirements of the communication partners. *High level policies* are exchanged and their intersection leads to a list of supported and required protocols for the connection. A choice can be made then, what the "best" protocol for this session will be.
In order to proof the concept we already implemented basic mechanism of ESAF for Linux in C++. The ESAF environment is still under development and misses central functionalities like mechanisms for key exchange or the use of certification authorities.
The next subsections will highlight the individual specialties of the different tasks of the ESAF approach.

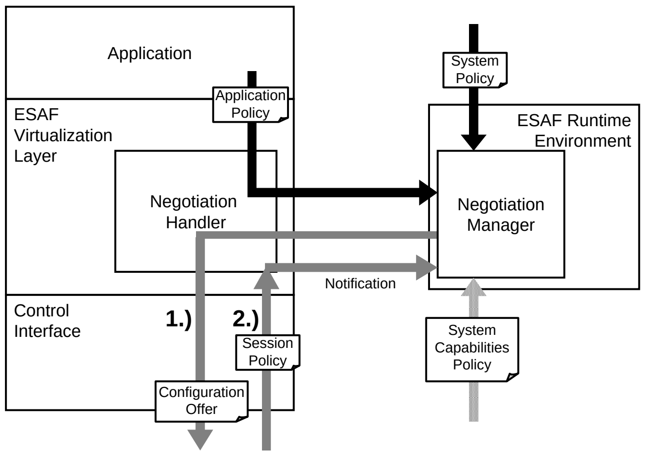

Fig. 2. ESAF Policy Processing/Traitement d'ESAF Policy

III.1 Architecture

The layered architecture was designed to achieve communication virtualization and configuration transparency for the application. Consequently the *ESAF Virtualization Layer* is the core of the framework. This layer is the generic communication interface for the application. It accepts *high level policies* as configuration requests and chooses autonomously appropriate communication setups. The application utilizes the *Generic Socket Interface* to carry out the communication then. The application is linked directly to the *Generic Socket Interface* contained in the ESAF library.
The Virtualization Layer uses internally the generic *Protocol Wrapper* interface. This wrapper also comprises a

generic interface and takes *system capabilities policies* for configuration. The wrapper allows the ESAF to easily introduce new protocols into the system without the need to recompile the ESAF library. The *system capabilities policies* allow to configure the protocols in depth while still being able to provide interchangeability of the particular protocols.
The *ESAF Runtime Environment* is designed as a daemon running constantly in the system. One functionality of the runtime is to make protocol configurations which require root privileges, for example of IPSec. Another aim is to keep the *ESAF Virtualization Layer* comparable lightweight and implement policy related functionalities here. Retrieval of the *high level policies* is such a functionality whereas the demon can keep track of the currently active security contexts.
The *Control Interface* is part of the application. The application is responsible for accessible ports from outside of the system and runs the control interface there. A remote host, willing to connect, initially negotiates a security context for the communication link, before data exchange can commence. The *Control Interface* allows the negotiation of security contexts during connection setup and the modifications of the context during runtime.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE esaf_high_level_policy SYSTEM "esaf_high_level_policy.dtd">
<esaf_high_level_policy>
  <security_requirements>
    <message_authentication>
      <minimum>5</minimum>
      <ideal>7</ideal>
    </message_authentication>
    <data_integrity>
      <minimum>7</minimum>
      <ideal>10</ideal>
    </data_integrity>
    <confidentiality>
     ...
    </confidentiality>
    <traffic_flow_confidentiality>
     ...
    </traffic_flow_confidentiality>
    <non-repudiation>
     ...
    </non-repudiation>
     ...
     ...
  </security_requirements>

  <communication_requirements>
    <connection_type>connection-oriented</connection_type>
    <reliability>reliable</reliability>
    <sequencing>yes</sequencing>
    <error_control>yes</error_control>
    <performance>
      <minimum>5</minimum>
      <ideal>10</ideal>
    </performance>
  </communication_requirements>

</esaf_high_level_policy>
```

Listing I. High Level Policy/High Level Policy

III.2 Requirements Description Language - RDL

We defined the ***R****equirements* ***D****escription* ***L****anguage RDL* to pass configuration requests along. The public interface of ESAF accepts *High Level Policies* whereas internally a *system capabilities policy* is used to describe the installed protocols.
We decided to use XML as a policy language, because it is easily extensible. Different versions of the ESAF can choose to ignore sections they do not understand. This is of course only possible if the section provides information marked as optional.

III.2.1 High Level Policies: It is important that these policies are truly protocol and configuration independent and describe the full range of requirements in a general manner. For such a policy language it is important to identify a set of language constructs and keywords that are able to express the full range of communication requirements. The security of a communication link is usually judged based on the degree it provides the following characteristics: authentication, integrity, confidentiality and non-repudiation. We identified two more parameters of high importance for secure communication: reliability and performance.

Security protocols are not equally optimized for all identified parameters. The level of security varies depending on key lengths and utilized encryption algorithms. The performance of the algorithm may also be an important factor, imagine a resource constrained device like a handheld computer. These thoughts led us to the decision to attach a scalar value to each service requirement to express the importance of the parameter on a scale between 0 and 10. The value 0 would mean "no importance" while 10 would give the parameter the highest priority. To differentiate even further we introduced the notion of *minimum* as a knock out barrier and *ideal* as the desired configuration value.

The security requirements are kept apart from the communication requirements in the policy. Inside the *security_requirements* element each parameter is stated with its *minimum* and *ideal* value. This element describes the typical security requirements as stated above. The tag *communication_requirements* encloses parameters like performance or reliability. Listing I depicts an example of such a high level policy.

High level policies which reside at the same system can be joined by the ESAF. The application specifies an application specific high level policy as well as the administrator can specify a system policy. These policies can easily be unified because they refer to the same system capabilities policy. The algorithm is easy, all minimum elements of the policy are compared and always the higher value is kept.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE esaf_system_ability SYSTEM "esaf_system_capability.dtd">

<esaf_system_capability>
  <supported_security_protocols>
    <security_protocol id="ipsec">
      ...
    </security_protocol>
    ...
  </supported_security_protocols>

  <supported_communication_protocols>
    <transport_layer>
    ...
    </transport_layer>
    <network_layer>
    ...
    </network_layer>
  </supported_communication_protocols>

</esaf_system_capability>
```

Listing II. Excerpt of System Capabilities Policy/Extrait du System Capabilities Policy

III.2.2 System Capabilities Policy: High level policies helped the stakeholders express their requirements. Now the question is how these high level policies get mapped onto particular protocols providing the desired properties. Our approach is to use *system capabilities policies*. These policies describe the available communication and configuration means of the system.
The basic elements of this format are the security protocols and the communication protocols. Here are the descriptions what a protocol can do and how it must be configured. If critical bugs in a security protocol are disclosed, the administrator can easily disable the corresponding entries in the *system capabilities policy* or degrade the security level. This will allow that the applications to use more secure protocols for their connections. The user will not even notice the change and the application does not have to bother.

```
...
<supported_security_protocols>
  <security_protocol id="ipsec">
    <supported_security_services>
      <confidentiality>
        ...
      </confidentiality>
      <message_authentication>
        <message_authentication>
          <auth_algorithm id="none">
            <key_length>0</key_length>
            <security_level>0</security_level>
            <performance>10</performance>
          </auth_algorithm>
          <auth_algorithm id="hmac-md5">
            <key_length>128</key_length>
            <security_level>2</security_level>
            <performance>9</performance>
          </auth_algorithm>
          <auth_algorithm id="hmac-sha1">
            <key_length>160</key_length>
            <security_level>5</security_level>
            <performance>6</performance>
          </auth_algorithm>
          <auth_algorithm id="hmac-sha2-256">
            <key_length>256</key_length>
            <security_level>8</security_level>
            <performance>3</performance>
          </auth_algorithm>
        </message_authentication>
          ...
      <non-repudiation>
          ...
      </non-repudiation>
        ...

    </supported_security_services>
    <required_communication_protocols>
      <!--protocols, which can be used with this security protocol -->
    </required_communication_protocols>
  </security_protocol>
  ...

</supported_security_protocols>
...
```

Listing III. Security Services in the System Capabilities Policy/Security Services dans le System Capabilities Policy

The *system capabilities policy* describes in detail the possible configuration options for each security protocol. We call policies at this detail level *low level policies*. These elements correlate directly with the elements of the high level policy. It is now possible to determine all possible encryption algorithms in the system which can provide a certain security service, for example, confidentiality.
Listing III shows an excerpt of the security section of an example *system capabilities policy*. Here are some supported configuration options for IPSec security services defined. This particular part shows some available encryption algorithms. Note how the key-length of 256bits for the AES algorithm increases the security level to 10 but decreases the performance level to 4. If implementations get more efficient or algorithms are considered less secure it is easy to change this policy to reflect the changes. The *security_protocol* tag can contain special information for each algorithm on how to configure the algorithm. In this example it is the *key_length* element.
The system must be aware of the dependencies between the different protocols. Each security protocol contains the section *required_communication_protocols* which determines in what combinations the protocol can be used.

III.2.3 Communication Interface: The *communication interface* provides abstraction of the actual protocols. Virtualization is reached by using *high level policies*. The interface itself must be general enough to allow the exchangeability of the underlying protocols but must must not limit the way a protocol can be used. The level of abstraction of the BSD socket interface[19] has already proved itself. The Socket++ interface[8], we chose to mimic, is an evolution of the BSD socket interface and tries to enhance the ease of use for programmers.
We added the method *negotiate_policy* to the interface for configuration by the means of *high level policies*. This method performs internally several steps to establish an agreement about the configuration of the communication link as described in the next section III.3. After the agreement is reached it establishes the communication with these parameters.

```
  class SecureConnection
  {
  private:
    void negotiatePolicy(const std::string &from, const std::string &to,const std::string &policy);
      ...
  public:
    //create secure connection
    inline SecureConnection(const std::string &from,const std::string &to, const std::string &policy) {
      ...
      this->negotiatePolicy(from, to, policy);
    }
    ~SecureConnection();

    //send data
    size_t sc = send(const char* data, size_t len);
```

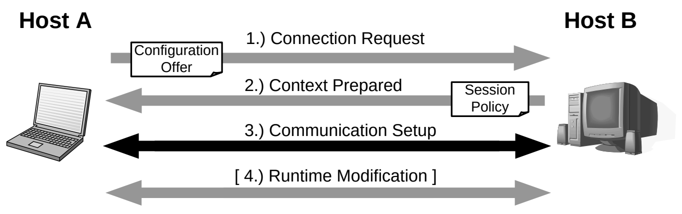

Fig. 3. Security Context Negotiation Sequence/Séquence de negotiation de contexte

```
    //receive data
    size_t rc = receive(char* data, size_t len);

    //disconnect
    void disconnect();

    //renegotiate the context of the connection
    void renegotiatePolicy(std::string* policy);

    //get context of the connection as xml
    std::string get_context() const;
    ...
};
```

Listing IV. An Excerpt of the Secure Connection Class/Extrait de la Secure Connection Class

The approach of providing a new socket layer provides the benefit that it does not matter at which layer of the protocol stack the required communication and security services are provided. It is even possible to replace the protocols during runtime and take for example, IPSec instead of SSL. The interface offers the method *renegotiate_policy()* for doing this.
The concept of using *high level policies* for configuration allows to extend the functionality of the framework without changing the interface. Applications must not be rebuilt to include these new functionalities. The extensible structure of the XML parameter will allow us to support *low level policies* in the future. These policies contain additional configuration options at the detail level of the *system capabilities policy*. One interface can be used then to provide loose or close control depending on the needs of the application.

III.3 Security Context Negotiation

When a connection has to be established it is necessary to perform a *security context negotiation*. The participants must agree on a set of possible protocols and a selection must then be made which protocols to use. At the moment ESAF supports only end-to-end communication for two participants.
Figure 3 shows the sequence of the negotiation. First a *connection request* must be made in step 1) by A. For this reason the ESAF at host A joins the high level policy of A with the system capabilities policy of system A. It tries to determine a set of protocols and configurations meeting the requirements. Only the entries which possess a security rating of equal or better then the minimum requirement specified by the *high level policy* will be included and form a special policy, the *Configuration Offer*. The generated Configuration Offer is now sent to host B inside the connection request. As an option the high level policy can be included to inform B about the *ideal* value for host A.

After host B received the request, it starts processing it together with its local policies. First, it must evaluate its own high level policy provided by its application and join it with its system capabilities policy to get the locally available configuration options. Then, the algorithm starts to determine the adequate configuration taking the minimum and ideal values into account. If the configuration is found the connection is prepared and a *Context Prepared* message is sent to A in step 2), containing the *Session Policy*.
Caching of the locally available configuration options can speed up the negotiation process. For hosts which must serve many similar requests it can save resources to implement a caching scheme at this stage.
In case host B is not able to find a possible intersection it will send an *Agreement Failed* message back to A

and attach its own high level policy and system capabilities policy. In the failure case host A could try to adapt its policies to find at least one possible communication link with host B. This modification should not be done automatically but through human intervention because it could lead to degradation of the security level.

After host A received the *Session Policy*, the connection setup of the protocol can start with the exchanged configuration information as shown in step 3). Furthermore, the application can always perform a runtime modification of the communication setup by renegotiating the parameters, as shown in step 4).
Of course, the ESAF middleware on host A must verify that the selected configuration is consistent with the request it sent in step 1). This must be done to detect manipulation attempts. However, it is a matter of mutual trust that the two hosts do not misuse their various decision options.

Authentication of messages is very important for the negotiation process to avoid manipulation attempts of the messages. Challenging is that at the time of the first negotiation possible authentication mechanisms are unknown. We decided to perform a sort of "optimistic authentication". The host A signs its initial request message with some authentication mechanisms which are likely to be supported by host B. Additionally, host A puts a nonce into the message which it expects to be included in the response from host B. Host B can proof the integrity of the message. When the response is received host A can detect replay attacks.

III.4 Support for Authentication and Trust

The virtue of ESAF is to decouple the protocol usage by a virtualization layer from the protocol instance and its configuration. However, certain security relevant information must be evaluated jointly by the application and the framework. Authentication is important in two respects, one is the level of trust ESAF puts into the authenticity of an entity and the other one is the provisioning of the identifier of the entity itself. A minimum required authentication level can be enforced by automatic means, connections below this thresholds will be blocked. The accomplished trust level can serve additionally as input for the application. The framework utilizes internally the strongest applicable authentication mechanism, for instance, if authentication via web of trust is rated with a medium confidence level the higher rated certificate based authentication with a trusted certification authority would be preferred. Yet the accomplished authenticity confidence level should be available to the application which can use it as a parameter for its authorization function.
Authorization is also an important issue for applications. Although an application could implement rule based authorization with the authenticated identifiers we provide an alternative functionality by offering a credential interface. A service request for a resource is tied to the provisioning of a particular credential. Access is solely granted if a valid credential is presented containing, for example, the certificate that an entity belongs to the group administrators which has special management commands at hands. Credentials must therefore be checked for authenticity and integrity themselves.

III.5 ESAF Components

ESAF Virtualization Layer: The virtualization layer ties the different components of the ESAF together. It communicates with the *ESAF Runtime Environment* running at the system and uses the *protocol wrapper* to configure and use the communication protocols. The external *control interface* takes care of the security context negotiations and renegotiations as described in section III.3. The *ESAF virtualization layer* is the central component which interconnects the different parts.

Protocol Wrapper: Abstraction is also an issue for the internal design of ESAF. It is important because the ESAF virtualization layer should be changed only seldomly. The wrapper encapsulates the specific interface of the protocol and its proprietary configuration options. It takes care to provide the functionalities of the standardized *protocol wrapper* interface by using the protocol specific functions. Low level policies are translated into the protocol dependent configuration process.

ESAF Runtime Environment: Since some functionalities can only be executed with system privileges the *ESAF Runtime Environment* was introduced. We use it in our implementation to perform IPSec configuration.
Other important functionalities of the ESAF Runtime Environment are to manage the *system capabilities policy* and to support the security context negotiation. When ESAF implements caching of security context negotiations, this will also be performed by the ESAF Runtime Environment.

Control Interface: As a protocol interface the *control interface* is exposed to any connection requests from foreign computers. It is a special server interface bound to a port to listen for *security context negotiation* requests. Here it is crucial to avoid critical flaws in the software design in order to prevent vulnerabilities like for example buffer overflows.

Negotiation Handler and Negotiation Manager: Several steps are necessary to conclude the *security context negotiation*. Responsible are two components: the *negotiation handler* which accepts connection requests and the *negotiation manager* which retrieves the system capabilities policy and implements the logic how to join high level and system capabilities policies.

III.6 ESAF Applications

ESAF provides for a large degree of abstraction from the communication internals. Yet, the choices how to use the API of the framework must not be limited in comparison to applications which support a particular communication mechanism natively. The ESAF Virtualization Layer must provide for the required flexibility.

```
...
//create ESAF Server at port
Secure_Server s= Secure_Server(address,true);

//listen
s.listen();

Negotiation_Handle* nh=0;
try {
  //accept one negotiation request
  nh=s.accept();
}catch(...){
...
}

//explicit negotiation of context
nh->negotiate(hl_policy);

//establish communication link with context
Secure_Connection* sconn=0;
sconn = nh->establish_conn(address);

//communication can now commence
...
```

Listing V. ESAF Server Example/Exemple Serveur ESAF

The establishment of a new connection requires two communication phases: One for the negotiation of the context and the other one for the communication itself. One important issue is how to handle incoming connections. The ESAF API mimics this semantic with the command *accept()* for incoming negotiation requests and the separate command *establish_conn()* for the setup of the communication link. The programmer of the application can choose how to handle the processing, for instance, by implementing a multi-threaded execution for the distinct phases. After the connection is established the Secure_Connection interface can be used comparable to a conventional socket.

```
...
Secure_Connection* s=0;
//establish connection to server_addr respecting the high level policy
s= new Secure_Connection(client_addr,server_addr,hl_policy);

//send some data
size_t l = s->send(buf,50);

//renegotiate the context
s->renegotiate(new_hl_policy);

//receive some data
buffer=new char[200];
```

```
l = s->receive(buffer,200);
...
```

Listing VI. ESAF Client Example/Exemple Client ESAF

The focus of the interface for clients is on the ease of use. One option is to establish the connection by simply creating a *Secure_Connection* object for a destination. The negotiation and the establishment of the connection happens transparently. If the client decides to use a different policy for a connection it can call *renegotiate()*.

IV Applicability of ESAF for autonomic configuration

In this chapter we want to demonstrate the ESAF middleware in action considering as an example the composition of a secure connection using IPSec. We demonstrate how the framework can select a configuration for the connection autonomously which matches the specified requirements. The scenario highlights how the different components work together, the messages are exchanged and the policies are applied.

Let us assume, that the user Alice wants to order a book via an online bookstore. She uses a web browser and wants to establish a secure connection with the purchasing system. Because the web browser and the purchasing service support the ESAF framework a secure connection can be setup:
We suppose that performance is an important factor for the purchasing system. Alice wants adequate security, therefore she specified her requirements at the GUI of the application. Internally these settings are translated into a ESAF High Level XML Policy. If she is an expert user she might modify this policy or even create the policy manually.
Eventually a policy like the one presented in figure I states the minimum security requirements for confidentiality, message_authentication, data_integrity and the communication requirements which are reliable, connection-oriented and error-controlled. We investigate the security service authentication in more detail for our example and assume that the level **5** is the minimum requirement for this element.

After the *high level policy* is defined the user can use her web browser to initiate the order. At first the web browser passes the high level policy of Alice to the local *ESAF Runtime Environment*. Here the *negotiation manager* becomes active and joins the high level policy with the *system capabilities policy* to generate the *configuration offer*. Some protocols and configurations which do not match the requirements of Alice are already discarded at this stage. To keep the example simple we consider only authentication algorithms and assume the other requirements to be equally fulfilled. The configuration offer contains three configuration options available at Alice's computer which must be rated:
The ESAF examines the first option, say the MD5 algorithm and discovers a security rating of **2** in the system capabilities policy. This rating is below the threshold of **5** and is discarded consequentially. Next, SHA-1 with 160 bits key length is rated with a **5** for authentication. The SHA-2 option with 256 bits is rated **8** for authentication. Because both algorithms match the requirement they will be included in the *configuration offer*.
Finally the configuration offer can be included in the *connection request* and be sent to the purchasing server.

Now, the configuration offer is received by the ESAF Control Interface of the purchasing server. Here the negotiation handler receives the offer and passes it together with the high level policy of the online bookstore to the *negotiation manager* inside the ESAF Runtime Environment.
This component has the complex task of joining three policies: the configuration offer of the web browser, the high level policy of the online bookstore and the system capabilities policy. First of all the high level policy of the bookstore gets joined with the local system capabilities policy. Again the protocols remain which fulfill the minimum requirements.
Now, these locally available protocols are joined with Alice's configuration offer. Because the purchasing server supports both SHA variants and they fulfill its requirements they remain in the result set. The ESAF Runtime Environment must decide which variant to choose. It assigns performance ratings of **6** for SHA-1 with 160 key length and **3** for the more secure SHA-2. Ultimately it selects the SHA-1 with 128 key length because it fulfills the minimum security requirements and achieves a better performance. Furthermore the ESAF selects TCP as a

compliant communication protocol, because the bank application and Alice allow only reliable, connection-oriented and error-controlled connections. The selected configuration set is put into a *session policy*.
Now, the purchasing server configures itself to accept the connection from Alicet's computer and sends the *Context Prepared* message including the *session policy* back to the web browser of Alice.

The ESAF at Alice's computer receives the message, and checks if the session policy is valid. It uses the policy then to establish the connection with the negotiated parameters. The protocol wrapper takes the session policy and establishes the chosen configuration of TCP over IPSec with SHA-1 and 160 bit key length. Finally, the purchase can be carried out.

V Conclusion

We presented a framework for autonomic configuration of end-to-end communication links. We described the concept of requirement driven configuration and how virtualization of the underlying mechanisms is achieved. Finally we demonstrated how applications can benefit from ESAF in typical scenarios.
The systems can act autonomously during communication largely because of the strict separation between requirements and configuration. Applications can leverage the virtualization to introduce new communication mechanisms and for hassle free utilization of well proven security services. The separation between requirements and particular configuration rules grants the flexibility to choose the "best" configuration option at a time. The selection depends on the requirements specified in form of policies and capabilities of the communication partners. The context of the link is defined during the negotiation phase. The first priority is fulfillment of security requirements; quality of service aspects are considered next. An ESAF application is unaware of the utilized mechanisms. Hence applications remain operable whilst new services are introduced into the system and deprecated ones are removed. Therefore, extensible security adaptation can foster the roll-out of innovative developments for large scale use.
However, ESAF is still under development. It remains future work to support mechanisms for key exchange and identity management. A thorough security analysis of the framework and a performance evaluation is under way.

References

[1] KEROMYTIS (A.). Some IPSec Performance Indications, 2001.
[2] KLENK (A.), MASEKOWSKY (M.), NIDERMAYER (H.), and CARLE (G.). ESAF - an extensible security adaptation framework, 2005.
[3] MUKHIJA (A.) and GLINZ (M.). CASA – A Contract-based Adaptive Software Architecture Framework, 2003.
[4] XU (C.) andGONG (F.), BALDINE (I.), HAN (L.), and QIN (X.). Building security-aware applications on celestial network security management infrastructure. In *International Conference on Internet Computing*, pages 219–226, 2000.
[5] SAXENA (B.). An adaptive security framework for wireless adhoc networks. *Wireless World Research Forum (WWRF)*, 2004. Euro-Labs.
[6] STILLER (B.), CLASS (C.), WALDVOGEL (M.), CARONNI (G.), and BAUER (D.). A flexible middleware for multimedia communication: Design, implementation, and experience. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, September 1999.
[7] IRVINE (C.), LEVIN (T.), NGUYEN (T.), and et al. Overview of a high assurance architecture for distributed multilevel security. *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security T1B2 1555 United States Military Academy, West Point, NY, 17Ű19 June 2002*, 2002.
[8] SWAMINATHAN (G.). C++ socket classes (1.12), 2004.
[9] GANZ (Z.) GANZ (A.), PARK (H.). Security broker for multimedia wireless lans: Design, implementation and testbed. 1998.
[10] The Open Group. Common Security: CDSA and CSSM, Version 2 (with corrigenda), 2000.
[11] KEENEY (J.). Chisel: A policy-driven, context-aware, dynamic adaptation framework, 2003.
[12] LI (J.), YARVIS (M.), and REIHER (P.). Securing distributed adaptation. *Computer Networks (Amsterdam, Netherlands: 1999)*, 38(3):347–371, 2002.
[13] LINN (J.). Generic security service application program interface, version 2. IETF, 1997.
[14] MAO (M.) and KATZ (R.). A framework for universal service access using device ensemble.
[15] YARVIS (M.). Challenges in distributed adaptation, 2000.
[16] YARVIS (M.), REIHER (P.), and POPEK (G.). A reliability model for distributed adaptation, 2000.
[17] FERGUSON (N.) and SCHNEIER (B.). A cryptographic evaluation of IPsec. Technical report, 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, 2000.
[18] YAHIAOUI (N.), TRAVERSON (B.), and LEVY (B.). Classification and comparison of adaptable platforms, 2004.
[19] LEFFLER (S.), JCKUSICK (M.), KARELS (M.), and QUARTERMAN (J.). The design and implementation of 4.3 bsd unix operating system. Addison-Wesley, 1989.

[20] NAQVI (S.) and RIGUIDEL (M.). VIPSEC: Virtualized and Pluggable Security Services Infrastructure for Adaptive Grid Computing, 2004.

[21] RAO (S.), FORMANEK (M.), and RIGUIDEL (M.). Prospect of new concepts in securing the cyberspace: Virtual paradigms, infospheres and pervasive computing, 2004.

[22] BRAY (T.), PAOLI (J.), SPERBERG-MCQUEEN (M.), MALER (E.), and YERGEAU (F.). Extensible Markup Language (XML) 1.0 (Third Edition), 2004.