ESAF - an Extensible Security Adaptation Framework

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

September 20, 2005

Abstract

The Extensible Security Adaptation Framework (ESAF) is designed to make configuration more flexible and avoid protocol-dependent application development. Among its features are the seamless integration of new protocols, exchangeability of corrupt protocols and utilization of the best protocol available for communication. This choice is based upon security policies specified by the administration, the user, and the applications. The security support is end-to-end and layer-independent. It also includes transport layer and quality of service requirements since the transport layer is transparent for applications using ESAF. High-level policies provide a fine-grained support for defining requirement levels. Requirements are therefore not binary, but scalar.

Keywords Security Adaptation, Virtualization, Security Context Negotiation, XML Security Policies, Quality of Protection.

1 Introduction

During the last decades the number of computers drastically increased as well as the demand for communication. This trend raises the issue on how to handle and guarantee security in those systems, considering their vast complexity. Traditional manual security management approaches reach their limits of applicability. Huge human resources are necessary to setup the systems to communicate securely and keep them running in the face of a constantly changing network environment. This administrative overhead makes it difficult to introduce new innovative services into the network and sometimes even impossible to use them at all.
As more and more computers are introduced into the network the problem gets even worse. Especially in ubiquitous scenarios where computers and other electronic helpers are embedded everywhere in the environment, it is not feasible to make a manual configuration for any communication and any device. The complexity and the unforeseeable number of possible interactions and communication interfaces require a self-configuration capability for the security and communication functions of the devices. A solution to this problem is offered by the virtualization of the communication mechanisms.

1.1 Technologies

During the last decade numerous security technologies evolved which are able to guard the user from attacks and intrusions. Security protocols like IPSec, SSL, TLS, SRTP are thoroughly evaluated but they require a careful configuration to guarantee security. Users without expert knowledge are often unable to make the right decisions and introduce severe vulnerabilities. Most large-scale networks that can be found in companies and universities are insecure due to misconfiguration of a few individual workstations. We propose a framework that can take the burden of making the right configuration decision away from the user and still offer experts the flexibility to tune specific configuration details.

1.2 The Configuration Problem

Protocols like IPSec suffer under the configuration complexity as Bruce Schneier[2] already stated in his survey on this protocol "Even though the protocol is a disappointment – our primary complaint is with its complexity – it is the best IP security protocol available at the moment." Hence, the challenge today is not to design a secure protocol but to configure a protocol to be secure!

Security configuration today can be done either at system level, for example by managing security policies or associations for IPSec, or at application level. Configuration at the system level is usually done to reflect the highest demand for security and therefore must use the protocol and cryptographic algorithm which offers the best security guarantees. The choice to use the "best" security protocol reflects only badly the true security needs at a given time. Which level of security is required depends a lot on what the user wants to do. During a session only certain data need strong protection. Other data like, for example, background images require less security. Another example for reduced security requirements is communication in trusted environments. The *security requirements* are volatile and change frequently even during the runtime of an application. It is obvious that security configuration needs to be done at a finer granularity and must be dynamic in contrast to the state today with static configuration at system level.

Some applications try to deal with the configuration of security protocols on their own and are therefore forced to support the protocol interface. Such applications break if old protocols become unavailable even if new security protocols are introduced into the system as a replacement. The ESAF deals with this problem by offering the application virtualization of communication. Virtualization makes it possible to introduce new communication and security protocols transparently. The application utilizes an abstract communication interface to hide the particular protocol. However, in case of ESAF the application can still control the communication context by specifying *high level policies*.

In contrast to the needs of applications an administrator prefers to have a single control point where a security policy can be specified and enforced. ESAF can provide this by using obligatory *system policies* which define the minimal security level. These system policies defined by the administrator must always be fulfilled for any communication. Application specific policies can only request higher security levels. As an option the ESAF framework can keep an audit of the communication contexts and inspect the applied configurations in detail. This audit helps to detect possible attacks and provides a mean to check the correctness of the decisions of the ESAF system.

The heterogeneity of today's network topologies and the vastly differentiating available security protocols at the end systems introduce additional challenges for protocol configuration. The prime task is to identify possible security configurations supported at the end systems. In a next step a consensus must be reached about the configuration that meets the requirements for the hosts best. This exchange during the *security context negotiation* must be sufficiently secured and designed with care to reduce the risk of vulnerabilities.

1.3 Structure of the document

The structure of the document is as follows. First, we present Related Work in Section 2. Then, the design goals which led to the development of the Extensible Security Adaptation Framework (ESAF) are discussed in Section 3. In Section 4 we finally introduce ESAF with its architecture and components. We deal with security considerations in Section 5 and finally summarize our approach in the Section 6.

2 Related Work

Several projects strive to provide flexibility for communication. We take a closer look at some research endeavors. The terminology of [16] helps us to categorize the systems.
The literature shows two main directions how to solve the adaptivity issue. One is to provide coordinated distributed adaptation functionality inside the networks like Yarvis proposed with the Conductor framework[23]. Conductor intercepts the communication and redirects it to the framework. Agents inside the network adapt the data streams according to a plan made by the Conductor framework at the data source. Although conductor provides some flexibility by altering the data streams it only provides rudimentary security services. Unspecified encryption algorithms are used to provide security services.
The other approach is to support the virtualization at the end-systems. The Generic Security Service Application Program Interface(GSS-API) [10] is standardized by the IETF and aims at providing a generic interface to use end-to-end security independent of the security mechanism and the communi-

cation protocol. The GSS uses tokens for the establishment of the security context and the protection of the communication. The application is responsible for the token exchange via a suitable transport protocol. GSS-API is well-established as an interface for authentication and key exchange. However the token based approach introduces a GSS dependent overhead for each message. Secure communication protocols, for instance SSL/TLS, cannot be accessed through this interface due to the peculiar message protection mechanism. Quality of protection can be achieved at the prize of loss of transparency of the underlying security mechanism. Furthermore, the API only covers security services and cannot provide for an abstraction of communication mechanisms. The communication and hence the context negotiation must be handled by applications itself.
The Common Data Security Architecture (CDSA)[4] is such an end-to-end cryptographic framework for creating security-enabled applications for client-server environments. The Framework implements its own security services at application layer to protect data transfers at run time. Hence it is not possible to use standardized, widely-used security protocols at different protocol layers. Policy based security configuration is not considered.
The Iceberg project[13] is similar to the Transformation, Aggregation, Caching and Customnization (TACC) architecture[3]. Thus it is a pure proxy system, specially designed for mobile devices with few resources. Proxy systems provide their services to legacy applications but fail to adapt to specific requirements of the applications. Secure protocols like IPSec or SSL are not supported and it is difficult to adapt the project to utilize these protocols.
The Chisel Framework[6] can be described as a reflective, application- and user-aware, end-system-based architecture. It is possible to specify adaptation policies to influence the adaptation process. Their reflective approach lets them change technical behavior without modifying their adaptation system. Security concerns were not in focus of this project and hence the framework fits only badly for the provisioning of security services.
In [18] Stiller introduced the DaCaPo++ system which uses protocol composition based on module chains. It is an application-aware system for the use on end-systems. Applications can influence their Quality of Service by selecting suitable protocols among a set of predefined protocols. This makes applications dependent on the supported protocols and future changes may cause problems with legacy applications. Security functionality is provided by proprietary security modules, thus security of the communication is difficult to proof because non standard protocols are utilized.
The Celestial Project[20] was developed at the North Carolina State University and is an application-aware and protocol-oriented security policy management system. The core component is the Security Management Agent(SMA), which performs the security protocols configuration and exchanges configuration messages with other Celestial nodes for connection setup. The SMA is implemented as a kernel module and offers additionally a simple socket interface for applications. SMNAs also reside at middleboxes on the data path and influence the connection setup by adding their own security policies to the message. Security services can be provided by those middleboxes and are therefore not end-to-end. Thus a high level of trust is required in the unprotected link with the middleboxes and of course the trustworthiness of the middleboxes must be assured. Celestial introduced a proprietary protocol called ISCP for the exchange of configuration messages and uses undisclosed algorithms and encryption functions for protecting the message exchange. The management of the security policies is done at the application level without an option for the administrator to influence or confirm the configuration. Therefor Celestial lacks the capability to enforce a system wide security policy. If a critical bug of a security protocol is exposed Celestial cannot easily adapt its policies.

3 Architecture Design Goals

While trying to provide communication and security virtualization it is important to state the requirements such a system must fulfill. Two basic principles are always visible in the design: a) be as autonomous as possible during connection establishment b) provide control at all stages and transparency of the state for the system.

We consider authentication to be out of scope for this document. That it can be provided with other means, say out-of-band or using a PKI infrastructure.

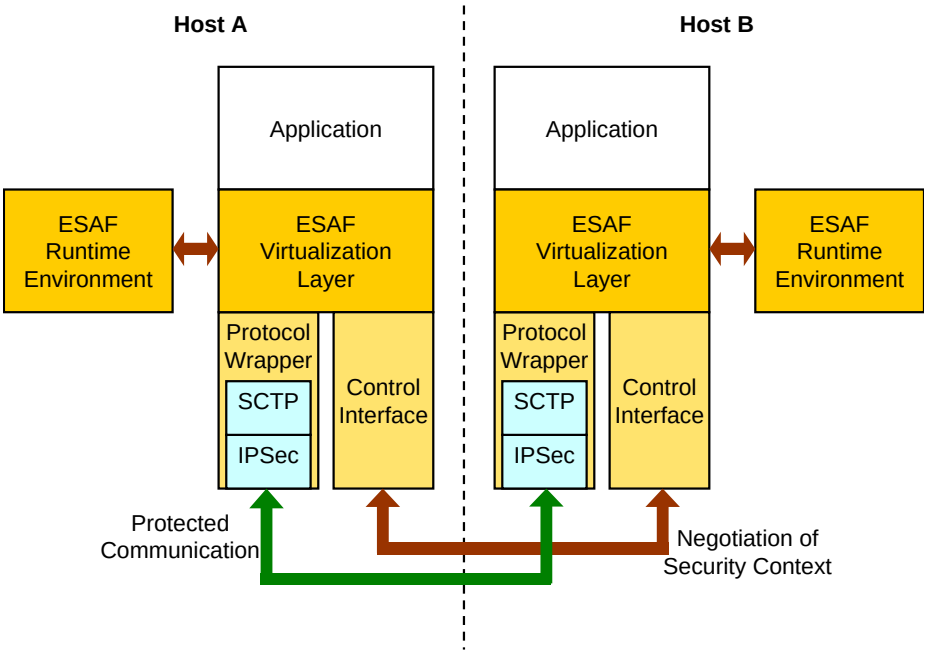

Figure 1: Extensible Security Adaptation Framework

- Self-Configuration: The communicating entities must find a configuration set that fulfills at least the minimal security and communication requirements of all participating entities. A consensus must be reached about which protocols and which configuration fits the requirements best. Self-configuration must be performed in a secure manner.

- Intrusion Handling: Security mechanisms sometimes comprise flaws and if these are disclosed a responsible administrator must react. Currently, one has to wait till an update becomes available to fix the flaw. It is usually no option to disable the protocol because a lot of applications would stop working. The mechanism must be able to disable protocols without harming the functionality of the system.

- Communication Virtualization: Communication interfaces must be abstracted and generalized to allow the exchangeability of underlying protocols. The abstract interface must provide all functionalities necessary for communication. It shall be possible to take control over the communication but must be able to work autonomously with self configuration.

- Context Adaptation: A secure perimeter requires maybe less security measures than a hostile environment. The computational resources that are available determine what kind of protocol is applicable at all. The context determines the communication requirements.

- Large Degree of Control: Different stakeholders take interest in the configuration of the connection. It must be possible for all participants to express their configuration demands. Administrators want to enforce a minimum security level whereas users take a large interest in the performance of the system. Applications can optimize the performance by adapting the security to the actually performed task.

- Ease of Control: The control must not only be possible but also be easy to implement. The method to express the requirements must be straight forward to formulate and to modify. The requirements

must be human understandable to allow administrators to make sure that the security context offers adequate protection. It must be possible to trace the state of the system during configuration for each message exchange.

- System Enhancement: Systems undergo many changes in the course of their lifetime. Protocols are added and configuration changes frequently. The installed applications should be able to take advantage of new system capabilities without being redesigned.

- Security: The architecture itself must be designed with security in mind. Possible weaknesses of the architecture must be detected to avoid introducing new security holes in the system.

4 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. It is not possible 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 a 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. For the present we made the assumption that the entities possess means to authenticate their communication partners.
The next subsections will highlight the individual specialties of the different tasks of the ESAF approach.

4.1 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

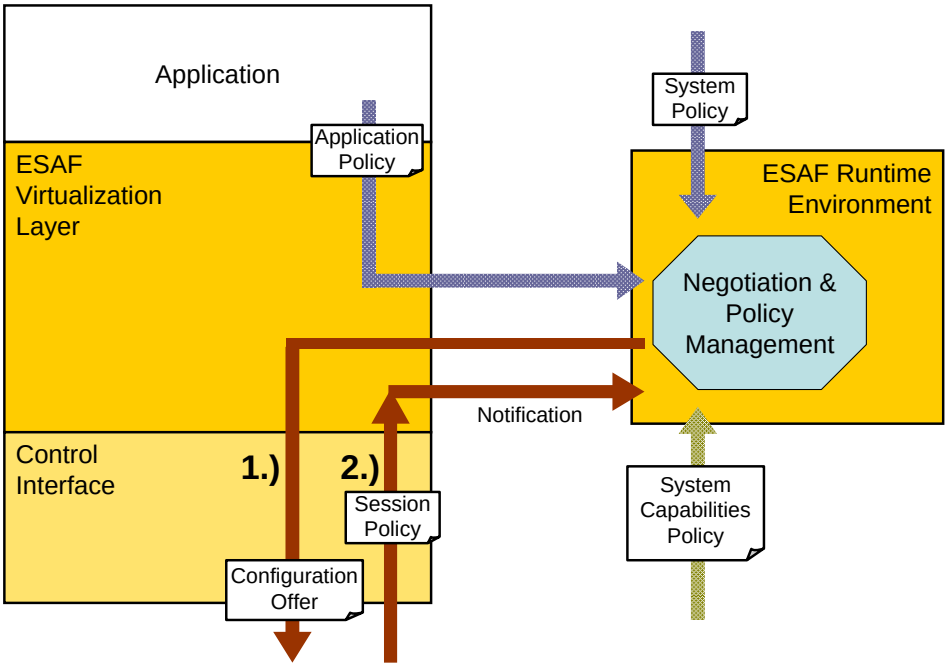

Figure 2: ESAF Policy Processing

configuration requests and chooses autonomously appropriate communication setups. The application utilizes the *Generic Socket Interface* of the ESAF library to carry out the communication then.
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. 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 daemon 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 1: High Level Policy

4.2 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.

4.2.1 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. This tradeoff between security and perfromance is also termed Quality of Protection(QoP) [5]. 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 1 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 dependent 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 straight forward, 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 2: Excerpt of System Capabilities Policy

4.2.2 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>
        <encr_algorithm id="aes128-cbc">
          <key_length>128</key_length>
          <security_level>9</security_level>
          <performance>6</performance>
        </encr_algorithm>
        <encr_algorithm id="aes256-cbc">
          <key_length>256</key_length>
          <security_level>10</security_level>
          <performance>4</performance>
        </encr_algorithm>
        <encr_algorithm id="twofish128-cbc">
        ...
        </encr_algorithm>
        ...
      </confidentiality>
```

```
      <message_authentication>
        ...
      </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 3: Security Services in the System Capabilities Policy

The *system capabilities policy* describes in detail the possible configuration options for each security protocol and a system local security rating. We call policies at this detail *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 3 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.

4.3 Communication Interface

The *communication interface* provides abstraction of the actual protocols. Virtualization is accomplished 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[8] has already proved itself. The Socket++ interface[19], 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 4.4. After the agreement is reached it establishes the communication with these parameters.

```
class Secure_Connection
{
private:
  void negotiate_policy(const std::string &from, const std::string &to,const std::
    string &policy);
  ...
public:
  inline Secure_Connection(const std::string &from,const std::string &to, const std::
    string &policy) {
    ...
    this->negotiate_policy(from, to, policy);
  }
  ~Secure_Connection();
```

```
    void send(const std::string &data);

    std::string receive();

    void disconnect();

    void renegotiate_policy(std::string* policy);

    ...
};
```

Listing 4: An Excerpt of the Secure Connection Class

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.

4.4 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 utilized security ratings 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 server 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*.

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.

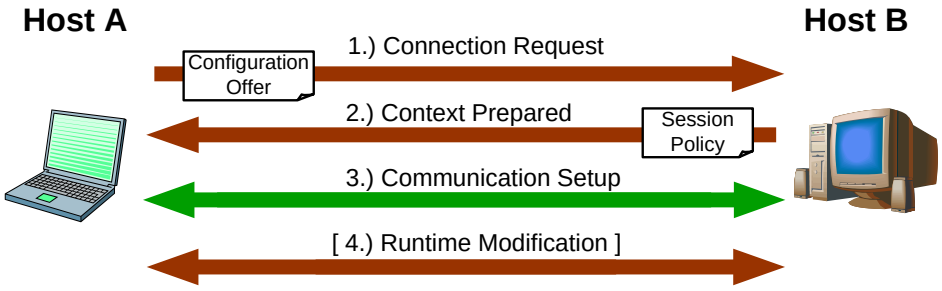

Figure 3: Security Context Negotiation Sequence

5 Evaluation

In this section we briefly evaluate the security aspects of the ESAF system. A thorough analysis and solutions to reduce possible problems are future work.

With respect to the security of the protocols that the system will use, we can state the following. ESAF does not use self-designed new security protocols, but uses well-known and well-evaluated protocols like IPSec or SSL and their implementations in current systems. This is not expected to be a weakness in ESAF.

More important for a future evaluation of ESAF is the analysis of the impact of and threats to ESAF itself. ESAF could be the point of attack. This could be due to weaknesses in the ESAF implementation. The *control interface* could be vulnerable because it accepts connections and could be misused for buffer overflows or DoS attacks. To limit this threat the *control interface* is active in the application and not in the runtime. Configuration weaknesses could be introduced by an attacker when she is able to modify low-level security policies system locally and propagate an insecure protocol as secure. Other possible problems could arise from the interaction of ESAF with security protocols.

We will discuss these points in detail and propose possible solutions in the future.

6 Conclusion

This document describes a framework for the virtualization of secure communication configuration called Extensible Security Adaption Framework. Applications using the framework are unaware of the utilized security mechanisms and the complex configuration thereof. They must only state their communication and security requirements and the ESAF will autonomously select and establish the best matching communication setup. Protocols can easily be introduced into the system or disabled if a critical vulnerability of a certain protocol is discovered. Because the ESAF virtualization hides the protocol stack completely from the application, it does not matter anymore at which layer security functionalities are provided.
Abstraction is reached by specifying two human-readable policy formats. One high level format describes the requirements whereas the other format describes particular protocols and configuration options at the system level. These XML policies are not only used internally but also for the connection setup. The parties wishing to establish a link exchange security policies and leave the connection setup up to the ESAF.
Although the implementation is under progress and the concept advances there are still open issues. First of all a thorough security investigation must be undertaken. Then issues like authentication and key exchange must be supported by this framework.
Our conclusion is that the ESAF approach can provide security virtualization and allows self-configuration. Of course, the benefits of ESAF are currently only available to applications which support the framework. However, the prospect of autonomous and secure self-configuration of communication is tempting.

References

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml) 1.0 (third edition), 2004.

[2] N. Ferguson and B. Schneier. A cryptographic evaluation of IPsec. Technical report, 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, 2000.

[3] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Symposium on Operating Systems Principles*, pages 78–91, 1997.

[4] T. O. Group. Common security: Cdsa and cssm, version 2 (with corrigenda), 2000.

[5] C. E. Irvine, T. E. Levin, and T. D. N. 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, 1719 June 2002*, 2002.

[6] J. Keeney. Chisel: A policy-driven, context-aware, dynamic adaptation framework, 2003.

[7] A. Keromytis. Some ipsec performance indications, 2001.

[8] S. J. Leffler, M. K. JcKusick, M. T. Karels, and J. S. Quarterman. The design and implementation of 4.3 bsd unix operating system. Addison-Wesley, 1989.

[9] J. Li, M. Yarvis, and P. Reiher. Securing distributed adaptation. *Computer Networks (Amsterdam, Netherlands: 1999)*, 38(3):347–371, 2002.

[10] J. Linn. Generic security service application program interface, version 2. IETF, 1997.

[11] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP). Internet Draft (draft-ietf-ipsec-isakmp-08), 1997.

[12] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories, Palo Alto, March 2002.

[13] R. H. K. Morley Mao. A framework for universal service access using device ensemble.

[14] A. Mukhija and M. Glinz. Casa – a contract-based adaptive software architecture framework, 2003.

[15] S. Naqvi and M. Riguidel. Vipsec: Virtualized and pluggable security services infrastructure for adaptive grid computing, 2004.

[16] N. L. Nesrine Yahiaoui, Bruno Traverson. Classification and comparison of adaptable platforms, 2004.

[17] S. Rao, M. Formanek, and M. Riguidel. Prospect of new concepts in securing the cyberspace: Virtual paradigms, infospheres and pervasive computing, 2004.

[18] B. Stiller, C. Class, M. Waldvogel, G. Caronni, and D. Bauer. A flexible middleware for multimedia communication: Design, implementation, and experience. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, Sept. 1999.

[19] G. Swaminathan. C++ socket classes (1.12), 2004.

[20] C. Xu, F. Gong, I. Baldine, L. Han, and X. Qin. Building security-aware applications on celestial network security management infrastructure. In *International Conference on Internet Computing*, pages 219–226, 2000.

[21] N. Yahiaoui, B. Traverson, and N. Levy. Classification and comparison of adaptable platforms, 2004.

[22] M. Yarvis. Challenges in distributed adaptation, 2000.

[23] M. Yarvis, P. Reiher, and G. Popek. A reliability model for distributed adaptation, 2000.