FFI-RAPPORT

17/01491

Cross-domain communication using an XMPP chat guard

–

Raymond Haakseth
Oddvar Brønstad (Thales Norway AS)
Øyvind Jonsson (Thales Norway AS)
Bengt Kristiansen (Thales Norway AS)
Nils Agne Nordbotten

Cross-domain communication using an XMPP chat guard

Raymond Haakseth
Oddvar Brønstad (Thales Norway AS)
Øyvind Jonsson (Thales Norway AS)
Bengt Kristiansen (Thales Norway AS)
Nils Agne Nordbotten

Norwegian Defence Research Establishment (FFI) 8 September 2017

Keywords

Informasjonssikkerhet
Informasjonsmerking
Sikkerhetsdomener
Datautveksling
Kommando og kontroll

FFI-rapport
FFI-RAPPORT 17/01491

Project number
1294

ISBN
P: 978-82-464-2940-3
E: 978-82-464-2941-0

Approved by
Nils Agne Nordbotten, *Research Manager*
Anders Eggen, *Director*

Summary

In current and future military operations the capability to communicate, distribute and share information is vital. Information superiority is achieved through the gathering, processing and sharing of data from sensors and humans. This requires that future information systems are interoperable and capable of sharing data and information with other systems. This includes instant messaging, also known as chat, which has become a popular alternative for informal message exchange between users.

Military systems have traditionally relied upon the use of physically separated security domains to provide confidentiality protection. While serving the purpose of protecting the confidentiality of information it also heavily restricts sharing of information. This includes information that otherwise could be shared.

A guard is an assured solution that may be used for connecting security domains. It protects the high domain from sharing information with the low domain that it is not allowed to share, i.e. information leakage. Guards inspect the confidentiality labels attached to the data in order to decide if it is releasable or not. It also contributes to the protection of the high domain from threats from the low side, like malware, thus protecting the integrity of systems.

This report presents a guard solution developed as part of the multilateral research project Coalition Networks for Secure Information Sharing (CoNSIS) II for chat messaging using the XMPP protocol. It enables users in one security domain to interact and exchange chat messages with users in another domain. The Chat Guard is designed and implemented in cooperation with Thales Norway AS. It reuses the basic architecture and design from the Mail Guard under development by Thales and the prototype XML/SOAP Guard developed in cooperation between FFI and Thales. Reusing the security critical components of these guards facilitates certification.

A prototype of the Chat Guard has been implemented by Thales Norway AS and tested. Through the testing it has been identified that the prototype may be too strict, stopping messages that are of use. Striking the right balance between protection and usability is important, and this report outlines how the finished guard may handle different types of messages. Also, lessons learned and experience drawn from the CD&E activity SMART on using chat in an operational scenario has been important input. The SMART initiative investigated whether the use of commercial smart technology, including chat messaging, could be used to provide situational awareness to units with little or no equipment today.

This work has shown that it is possible to design and implement a guard for chat using the XMPP specification based on the existing guards in development. A working prototype has been established that may be developed into an operational system. The Chat Guard is designed with an aim of Common Criteria EAL 5 certification.

Sammendrag

Evnen til å kommunisere, distribuere og dele informasjon er avgjørende for nåværende og framtidige militære operasjoner. Informasjonsoverlegenhet oppnås gjennom å samle, prosessere og dele data fra sensorer og mennesker. For å oppnå dette må framtidens informasjonssystemer være interoperative slik at informasjonen ikke er bundet til ett system. Det finnes en mengde forskjellige militære kommunikasjons- og informasjonssystemer som brukes for å utveksle informasjon. Lynmeldinger, også kjent som chat, har etter hvert blitt et populært alternativ for uformell meldingsutveksling mellom brukere.

Militære systemer har tradisjonelt brukt fysisk skilte sikkerhetsdomener for å beskytte konfidensialiteten til både systemer og informasjon. Dette gir konfidensialitetsbeskyttelse, men samtidig er det også et stort hinder for deling av informasjon. Dette inkluderer informasjon som ellers kunne vært delt, men som ikke kan deles fordi den er lagret eller behandlet i et system i et annet sikkerhetsdomene.

For å knytte to sikkerhetsdomener sammen brukes gjerne en såkalt guard-løsning. Guarden beskytter det høye domenet mot informasjonslekkasjer (konfidensialitetsbeskyttelse) ved å stoppe informasjon som ikke skal eller kan deles med lavere domener. Dette gjøres ved at guarden inspiserer konfidensialitetsmerker som er påført informasjonen og som beskriver sensitiviteten. Hvilken informasjon som kan frigjøres, er avhengig av hvilken policy guarden er konfigurert med. Guarden eller omkringliggende mekanismer må også sørge for at skadevare som virus og andre dataangrep ikke får passere (integritetsbeskyttelse).

I denne rapporten presenterer vi en guard-løsning kalt Chat Guard, som kan brukes for lynmeldinger som bruker XMPP-spesifikasjonen. Denne guarden gjør det mulig for en bruker i et sikkerhetsdomene å utveksle lynmeldinger med brukere i andre domener, samtidig som konfidensialiteten til informasjonen og integriteten til systemene er beskyttet.

Chat Guard er et resultat av samarbeid mellom Thales Norway AS og FFI og har vært en del av det multilaterale forskningssamarbeidet Coalition Networks for Secure Information Sharing (CoNSIS) II. Denne guarden gjenbruker både arkitektur og design fra Mail Guard som er under utvikling av Thales, og fra prototype XML/SOAP Guard som ble utviklet i samarbeid mellom FFI og Thales. Gjenbruken av sikkerhetskritiske komponenter bør gjøre evaluering og sertifisering enklere. Thales Norway AS har stått for implementering og testing av Chat Guard-prototypen.

For alle sikkerhetsfunksjoner er det viktig å finne balansen mellom å beskytte og samtidig være brukervennlig. Testing av prototypen har vist at flere viktige meldingstyper ble stoppet. Vi har også brukt erfaringer fra CD&E aktiviteten SMART som ble gjennomført av FFI i 2016, som blant annet undersøkte bruk av lynmeldinger i et operasjonelt scenario.

Denne rapporten viser at det er mulig å lage en guard-løsning for lynmeldinger med tilstrekkelig høyt tillitsnivå. Det er også laget en funksjonell prototype som kan utvikles videre til ferdig produkt. Chat Guard er designet med tanke på evaluering til tillitsnivå Common Criteria EAL 5.

Content

1 Introduction

The capability to communicate, distribute and share information is vital for both current and future military operations. A range of communication and information systems are available for the warfighter in order to enable this information sharing, ranging from voice communication via military messaging to formal command and control systems. The latter years, instant messaging, also known as chat, has become a popular additional communication channel that is more informal, lightweight and easy to use than formal military messaging. Military systems are also moving from self-contained stove-pipe systems to more modular designs that are able to utilize information from different sources more effectively and thus maximizing the operational effect.

Military information and communication systems have traditionally relied upon the use of physically separated security domains to protect confidentiality. This separation prevents information from leaking from a higher sensitive system to a lower sensitive system, and the different security domains are not allowed to be interconnected. The downside is that it also makes it difficult to share information that otherwise could be shared between users and systems in different security domains. The lack of information sharing makes it difficult to take the full advantage of the modern information systems.

As chat becomes more used as an important communication channel it also becomes important to provide cross-domain chat. Users of information systems have the need to communicate and send chat messages to users in other security domains. Enabling this requires the implementation of mechanisms that supports this interaction while at the same time ensuring that the security of the systems and security domains involved are maintained. Typically this involves preventing information leakage (protecting the confidentiality) and preventing transfer of malware and other cyber-attacks. Since these mechanisms are used to interconnect different security domains it becomes vital that they perform as intended. A high assurance level is thus required.

This document describes the development of a prototype chat guard that can be used to enable chat messages between users in different security domains. The design and implementation of the chat guard has targeted a CC EAL 5 evaluation. The work described here includes functional description and evaluation of restrictions posed by security considerations, as well as requirements on chat clients and servers.

This work has been a cooperative effort between the Norwegian Defence Research Establishment (FFI) and Thales Norway AS. The work has been performed as part of the multinational research collaboration Coalition Networks for Secure Information Sharing (CoNSIS) phase II. CoNSIS II consists of four tasks: (1) Communication Infrastructure, (2) Information and Integration Services and Functional Services, (3) Cyber Security, and (4) Service Management & Control. This work has been part of Task 3 Cyber Security.

This report is organized as follows; in Section 2 the Chat Guard is presented including a functional description, design considerations and requirements. Section 3 describes how the Chat Guard was tested and demonstrated. A discussion on the findings and lessons learned from implementing and testing the prototype is presented in Section 4 and the report is concluded in Section 5.

2 Chat Guard

The Chat Guard is based on the architecture developed for the prototype XML/SOAP Guard [1][2], which in turn is based on the architecture of the Thales Mail Guard.

Chat is a collaboration tool used to coordinate information that does not necessarily need to be archived etc. It is typically less formal than information conveyed as formal messages. The Chat Guard allows instant messages according to XMPP to flow between two security domains if and only if the messages comply with the Guard policy. All messages must be signed and labelled with their classification.

XMPP is an open standard for messaging and presence. It was standardized through the Internet Engineering Task Force (IETF) in 2004 and revised in 2011 with the publication of RFC 6120 [3], RFC 6121[4] and RFC 7622 [5]. Extensions to the core specifications, known as XMPP Extension Protocols (XEP), are governed and published by the XMPP Standard Foundation. This is an independent, nonprofit standards development organization. Since it is an open standard anyone is free to contribute with extensions. XMPP is a proven technology for messaging and is used extensively. The fact that XMPP is both a standardized and proven specification for messaging makes it a good choice for the Chat Guard.

2.1 Functional description

The basic function of the guard is to reject all incoming objects unless they comply with a configured set of rules. The main rules are to verify that the security label is within configured limits, and that the object is correctly signed. In general, the guard will trust data that is covered by a validated signature. Other data is in principle not trusted, but can be accepted depending on the guard policy (possibly after filtering or other checking). The guard requires that external Border Protection Devices are present, allowing network-based threats to be detected and countered outside the guard.

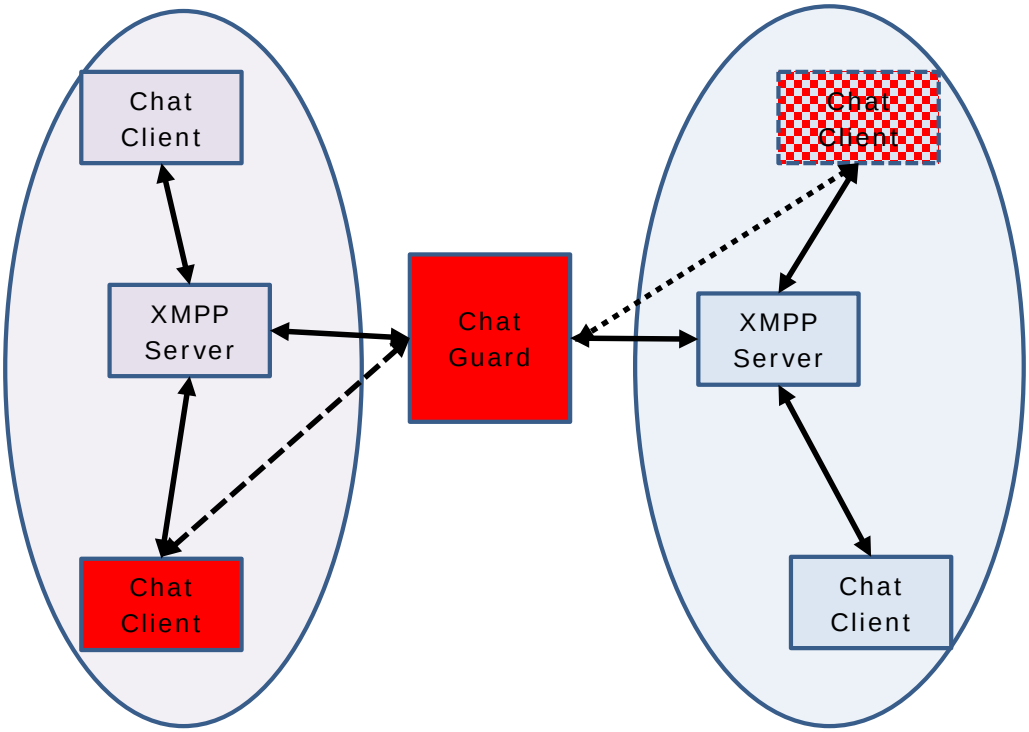


Figure 2.1 Chat Guard between two security domains

The Chat Guard interconnects two security domains as shown in Figure 2.1. It allows XMPP server to server traffic and it plays the role as a proxy XMPP server. The TCP connection from one XMPP server is terminated in the Guard and a new connection is set up towards the XMPP server on the other side. I.e. the logical connection between the two XMPP servers is maintained through the Chat Guard. XMPP chat clients cannot initiate a connection directly to a server in the other domain as only server-to-server connections are allowed through the Chat Guard.

The capabilities that are specific to the Chat Guard are identified as follows:

- XMPP objects (Stanzas[1]) are transported inside streams over TCP
- XMPP objects (Stanzas) are labelled using an XML Confidentiality Label [6]
- XMPP objects (Stanzas) are digitally signed using a signature that is bound to the label according to the XML Binding Profile [7]
- Signed parts of the XML object are normalized and all information is kept within one Stanza
- XML Binding and Signature information is carried within a well-defined header field
- Access control is based on signatures and Addressing Services [8]

[1] In XMPP, stanza is the basic unit of communication (similar to a packet or message). Stanzas can be of type message, presence or iq.

The Chat Guard configuration is defined by a Configuration Vector containing all relevant configuration attributes.

2.2 Basic guard design

The design of the Guard is aligned with edition 1.3 of the “Protection Profile for the NATO High-Assurance ABAC Guard” [9] . The HAAG PP defines an overall security design, and specifies a set of security functions. The Guard implementation is targeted at a Common Criteria EAL 5 evaluation.

The Guard is designed around a MILS type separation kernel, providing trusted mechanisms for separating different parts of the guard from each other. This allows the guard design to implement the principle of ”least privilege” in a way that keeps different security aspects separate, aiming to simplify the security evaluation.

The Chat Guard builds on the previous XML/SOAP Guard prototype, in turn an extension of the Mail Guard architecture. All guards build on the same security principles and the same overall design. The Chat Guard is implemented on the target operating system running on a standard PC platform. An abstraction layer was created to facilitate porting to the target platform.

The general Guard structure is shown in Figure 2.2.

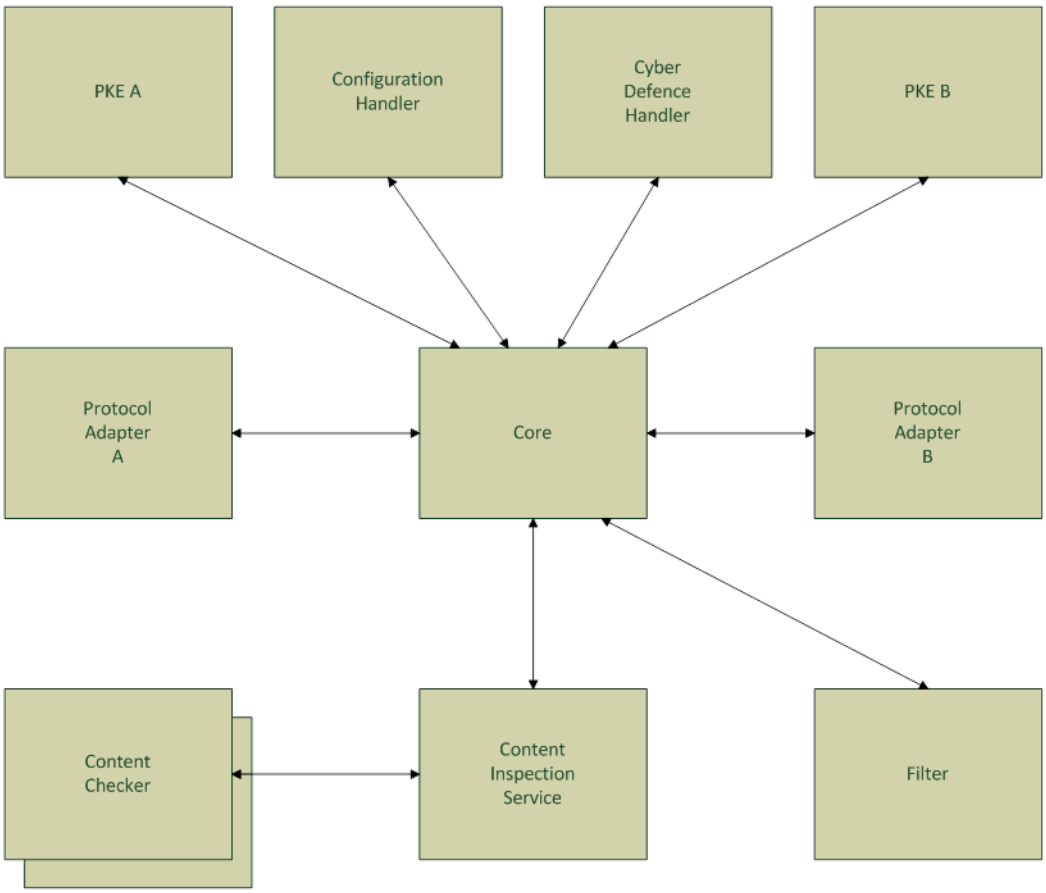


Figure 2.2 Guard design

The Guard design is based on having a set of services that as far as practical are independent of the communication protocols used. The specifics of each protocol are handled in the Protocol Adapters towards the connected systems.

The individual functional blocks have the following overall responsibilities:

- *Protocol Adapter A and B*
  There is one set for each protocol (STANAG 4406 and SMTP respectively for the two Mail Guards, XMPP for the Chat Guard and SOAP for the XML/SOAP Guard), performing the protocol handshake, mapping protocol attributes to the attributes used for filtering, and handling error situations (including objects being rejected by the Guard).
- *Core*
  The Core ensures that each object submitted to the Guard is correctly processed. This includes subjecting the object to Content Inspection Services appropriate to the type of object (e.g., XML validation), mapping attributes to avoid information leakage (e.g., identifiers and addresses that have a scope within one of the connected networks), activating the Filter, and dispatching the needed sub-functions in the correct order.
- *Content Inspection Service*
  The Content Inspection Service provides an interface mechanism for plugging in various types of Content Checkers. Default Content Checkers include dirty-word checking and XML Schema Validation. Other application-specific checkers can be added, allowing a secure mechanism for inspecting and/or validating an object, without risking modification or leakage of the object content.
- *Filter*
  The Filter performs the Guard filtering function, determining whether a specific object is allowed to pass the Guard. Filtering decisions are based on configurable criteria (supplied by the Configuration Handler, possibly modified via the Cyber Defence Handler), and uses functionality from the Core and PKE.
- *PKE (Public Key Enablement) A and B*
  There is one set for each connected system, performing digital signature validation and generation, and certificate and CRL validation. The PKE can be interfaced to different PKI systems.
- *Configuration Handler*
  The Configuration Handler provides an interface for configuring the Guard with one or more Configuration Vectors, each defining a policy for which messages are allowed to pass the Guard. Each Configuration Vector includes data such as network interface parameters, selected content filters, XML Schema, security label range, etc.
- *Cyber Defence Handler*
  The Cyber Defence Handler provides an interface for monitoring and controlling the Guard, e.g. by changing to a different Configuration Vector as a result of reported status information.

2.3 Chat guard design

In order to support Chat messages (transported inside XML objects), the base Guard design had to be extended. This applies to:

- *Protocol Adapter*
  A protocol handler for XMPP was added. The XMPP Stanza attributes are extracted and given to the Filter.
- *Core*
  The mapping function for identifiers/addresses was extended to support a more general way of encoding, e.g., the attribute type syntax used by XML documents. For identifiers, this applies to IDs in message, presence and IQ stanzas (to associate a response with the correct request).
- *Filter*
  The filter function was updated to use plain text security labels (similar to the XML/SOAP Guard).
- *Configuration Handler*
  Configuration data was added for XMPP (e.g., XML schema definition).
- *Content Inspection Service*
  An XML Validator could be added as a content checker. Note that this requires stanzas to be self-contained, i.e., a single Chat message cannot be transported in multiple Stanzas.

2.4 Chat guard protocols

The Chat Guard forwards traffic between two XMPP servers within the respective domains. The prototype has been implemented using the following protocols and specifications.

- *Streams*
  The XMPP Core specification [3] defines the relation between Streams and Stanzas. An XMPP stream is a persistent connection used for exchanging XMPP stanzas between two XMPP entities. The initial Stream negotiation is unencrypted. An encrypted Stream is negotiated over the initial Stream, which is dropped once the encrypted Stream is opened. The Chat Guard will only accept “instant messages” and “responses”. The set of acceptable stanzas is predefined.
- *Security*
  The Chat Guard requires TLS encrypted Streams. Unencrypted Streams are only accepted for the initial negotiation of encryption keys. This negotiation is performed according to the STARTTLS specification [10].
- *Authentication*
  SASL (simple authentication and security layer protocols) negotiation (RFC 4422 [11]) is supported as the authentication mechanism.
- *XML stanzas*
  After stream negotiations the servers may exchange <message/> stanzas. The following attributes are supported:
  - “To” attribute (in server to server streams)
    The “To” attribute must be an XMPP address. It is mandatory between servers (auto broadcast via this interface is not accepted for presence or IQ without «To»).
  - «From” attribute (in server to server streams)
    The “From” attribute must be an XMPP address.
  - “Id” attribute (in server to server streams)
  - “Type” attribute (in server to server streams)
  - “Xml:lang” attribute (in server to server streams)
- *XML elements*
  In addition to the stanza attributes above, the XML elements are supported for confidentiality label [6] and signature [7].

2.5 Chat Guard requirements and restrictions

When using the Chat Guard some additional requirements and restrictions to the general XMPP specifications apply. These requirements and restrictions have the aim of protecting the Chat Guard from leaking information. They are based on the experience drawn from the development and operational use of other guards and from the implementation and testing of the prototype

Chat Guard. Differences between these requirements and the implemented prototype exist and are noted in the text.

2.5.1 Message

If the Chat Guard is configured to filter messages according to labels the XMPP message must contain a Confidentiality label according to the NATO XML Confidentiality label standard [6] and be bound to the message according to the NATO Metadata binding standard [7].

2.5.2 Presence

Presence is a broadcast or publish-subscribe operation. Presence is a security sensitive service, and it must be carefully considered how it should be allowed through the Guard. It should be constrained to only allow specific entities to publish presence. Both sides of the Guard will have a list of entities which are allowed to send out presence. The lists should be configurable, and may be different in each direction. This is further discussed in section 4.2.1

Presence stanzas were discarded by the Chat Guard prototype.

2.5.3 IQ

IQ is a request-response operation and therefore a security sensitive function. A minimal list has to be worked out to list IQs (with response) which may be accepted and the responses that are allowed in response. The list should be configurable, and may be different in each direction. IQs must also be signed according to the Metadata binding standard [7]. IQ messages and responses are further discussed in section 4.2.2

IQ stanzas were discarded by the Chat Guard prototype.

2.5.4 Streams

The Chat Guard always operates on a server to server stream (i.e. direct client-server interactions are not supported through the Guard). The stream must be secured by TLS and authenticated by SASL.

2.5.5 Confidentiality label

The XML Confidentiality Label is implemented according to the NATO Standard "Confidentiality Metadata Label Syntax" (ADatP-4774) [6]. The confidentiality label must represent the complete object.

2.5.6 Multi-User Chat (Chat rooms)

Multi-User Chat (MUC), also known as Chat rooms, were not supported by the prototype. A discussion on how chat rooms can be supported is found in section 4.3.

2.5.7 Error responses

Detailed error responses must be restricted. A list must specify the error responses that are allowed to pass the Chat Guard. Non-accepted error responses must be anonymous.

2.5.8 Digital signatures

The Digital Signature must cover the entire XML object (excluding the signature itself). The digital signature is implemented according to the W3C recommendation "XML-Signature Syntax and Processing (2008)"[2].

The signature validation and generation uses the same interfaces and mechanisms as the Mail Guards and the XML/SOAP Guard. This implies that the specific algorithms and key lengths can be adapted to the specific usage, and functions that reference the private key will be performed by a Hardware Security Module (HSM). For the Chat Guard prototype, the SHA-256 hash algorithm and the RSA encryption algorithm was selected.

2.5.9 XML

XML objects must be presented in a standard form before digital signatures can be validated. This canonicalization process is implemented as follows:

- Within the Reference element, the "inclusive, without comments" strategy is used. The reference covers the entire XML object, and thus there are no "parent" element and no inherited namespaces. This is the default algorithm, and is not identified in the XML object.
- Within the SignedInfo element, the "exclusive, without comments" strategy is used. This avoids duplicated namespaces in the released XML object. The specific algorithm is

<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

Outgoing XML objects keep the canonicalization applied on reception.

2.5.10 Certificates and Certificate Revocation Lists

Certificates and Certificate Revocation Lists (CRLs) are accessed and used in the same way as the other Guards, i.e. via the PKE-A and PKE-B functions shown in Figure 2.2.

For the experimentation and demonstration (see Chapter 3), the Chat Guard prototype used X.509 version 3 certificates and version 2 CRLs. End user certificates were signed using SHA-256 and RSA (with 2048 bit key length), and the end user public key was an RSA 1024-bit key. Specific algorithms and key lengths are adaptable to the specific usage scenario.

[2] http://www.w3.org/TR/2008/PER-xmldsig-core-20080326

2.6 Platform

The Chat Guard demo is running on a MILS type separation kernel on a Personal Computer. The Border Protection Devices are omitted in this demonstration scsenario.

2.7 Use cases

Figure 2.3 shows how the Chat Guard can be used. There are at least two different scenarios:

- Scenario #1
  Typical usage is to have a labelling service in the high domain and no labelling service in the low domain. All messages from the high domain must be labelled in order to allow the filtering function to decide if they are allowed to be released into the low domain.
- Scenario #2
  Labelling service in both domains.

When security labels are used, they can be added by the Chat Client or by a special-purpose labelling XMPP Server. Such an XMPP Server could perform labelling for a group of Chat Clients in a security domain.

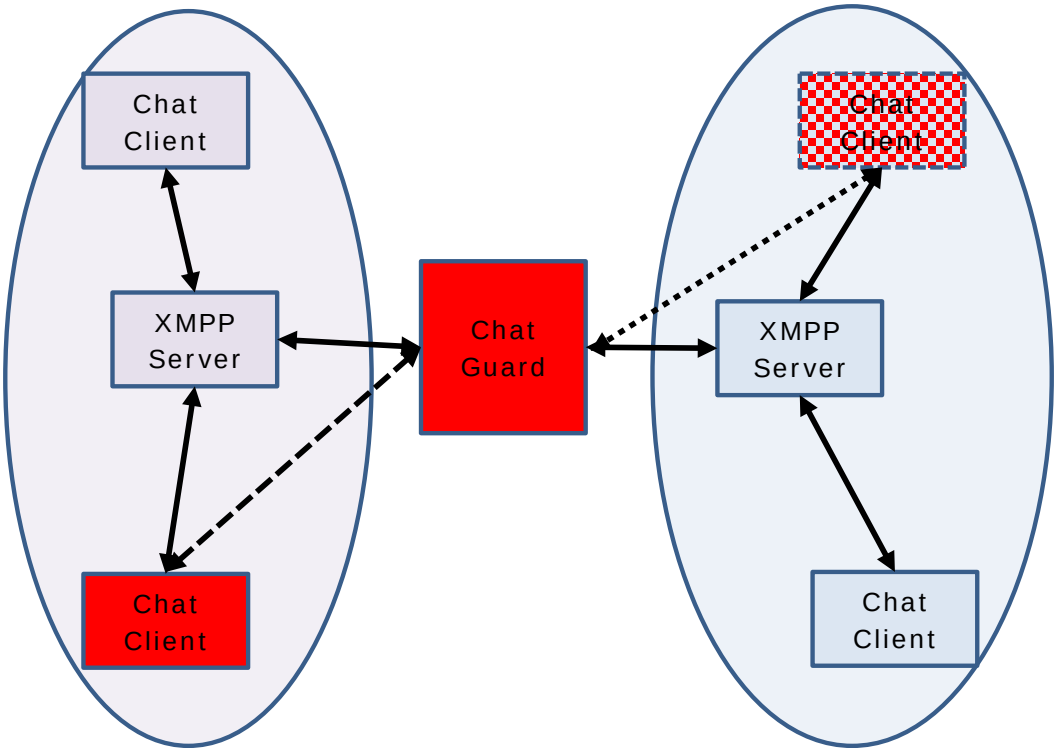


Figure 2.3 Chat Guard use case

2.8 XOchat – test client

A prototype Chat Client was developed in parallel with the Chat Guard as a way to test the Chat Guard functionality. This client provides a subset of functionality for sending and receiving chat messages, and for retrieving the contact list from the XMPP server. In addition, the test client adds the capability to mark chat calls with confidentiality labels, and to digitally sign chat messages.

3 Guard testing and demonstration

Testing and verification has been an important activity in the Chat Guard development. During development Thales Norway AS has performed testing according to their development standards. This includes establishing an on premises testbed. This testbed was also replicated at FFI, outside the development environment. Finally the solution was demonstrated in a close to operational setting and scenario.

3.1 Generic Chat Guard testbed

The generic testbed established at FFI consists of two simulated domains, one high and one low, which are interconnected with the Chat Guard, see Figure 3.1. Each domain has its own Chat Server. Users have a chat account registered with the chat server in their own domain, known as the home server. A user is identified with the combination of username and home server. For instance the user Bob at the server HighDomain is identified as Bob@HighDomain. A user exchanges messages with its home server only, which relays messages to the correct recipient. The chat interaction is never directly between two users. If a message is addressed to a user in another domain the chat server forwards the message to the home server of the recipient.

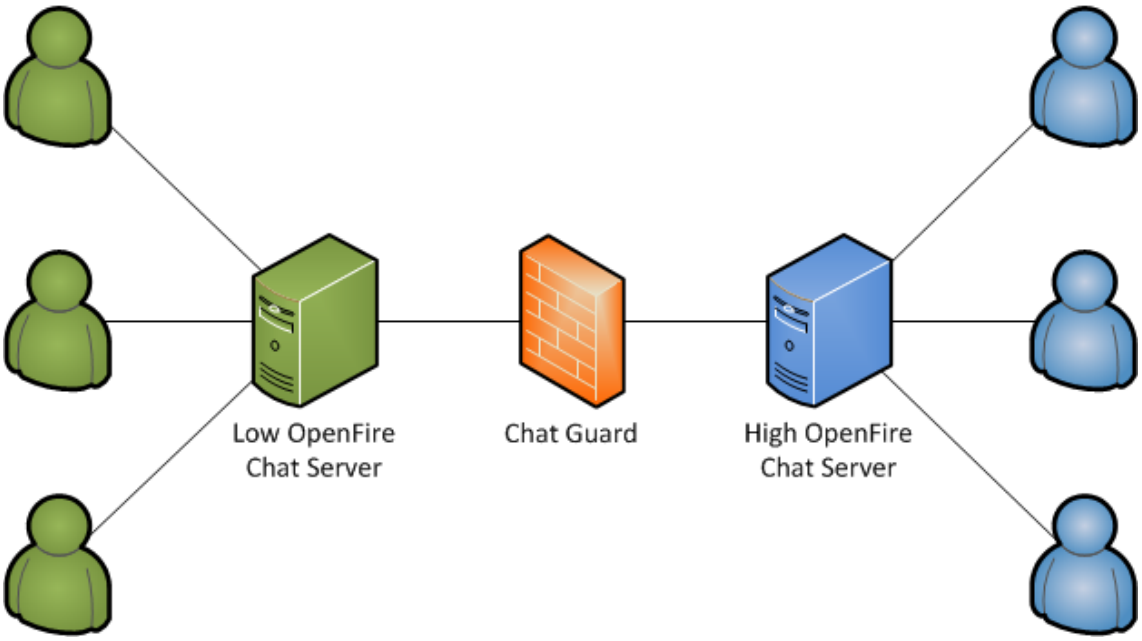


Figure 3.1 Chat Guard testbed architecture

The Chat Guard only handles XMPP server-to-server connections, all other connections are disallowed. Thus users cannot use a chat server in another security domain as home server. A secure server-to-server session is setup between each of the two chat servers and the Chat Guard. In essence the Chat Guard can be viewed as a transparent proxy between the two chat servers. The chat server believes that it is communicating directly with the receiving server while it is actually communicating with the Chat Guard. All communication between the chat servers and the Chat Guard was secured using SASL (Simple Authentication and Security Layer) and TLS (Transport Layer Security).

The testbed does not have any DNS capability.

3.1.1 Chat Guard configuration

The Chat Guard interconnects and enables chat messages to be exchanged between two security domains. A description of the general functionality of the Chat Guard can be found in section 2. This section describes the specific configuration used at the FFI testbed.

The Chat Guard has five different network interfaces. In the testbed three of these were used, two for connecting the guard to the different security domain networks and one for management. The remaining two interfaces are used for connections to PKI systems, one in each domain. For simplicity this was however not used in the testbed. The three network interfaces used were configured to have IP-addresses within the range of their respective networks. The IP address of the XMPP server within a domain is also configured and stored in the protocol handler for the given domain. This makes it possible for the protocol handler to

forward messages to the correct XMPP server. Also configured within the protocol handler is the domain name of the XMPP server.

All connections to the Chat Guard must be authenticated using the SASL protocol. The chat servers expect the other side to identify itself using a certificate with a common name equal to the xmpp-domain name of that server. Since the protocol handlers acts as the XMPP server in the opposite domain it must also be configured with a certificate with a corresponding common name. The corresponding private key must also be available for the protocol handler.

The Chat Guard can be configured to release different classifications of information dependent on the scenario and the security domains that are connected. The default configuration in the testbed is to release all messages labelled Norwegian unclassified to confidential, NATO Unclassified to Secret and unmarked messages. Messages with confidentiality label with value Norwegian SECRET or higher and labels with value higher than NATO SECRET are thus disallowed and stopped by this configuration of the guard.

3.1.2 XMPP server configuration

The testbed used the open-source Openfire[3] XMPP server from Igniterealtime as XMPP chat servers on both the low and high side. No modifications were done to the servers except configuration, most notably configuration of certificates, users and manipulation of IP addressing. In addition basic XMPP configuration was performed, for instance adding an xmpp-domain name.

Both the low and high side XMPP servers were configured to use the SASL and TLS protocol when connecting to other servers. They were also configured to require the use of SASL and TLS when receiving connections. A certificate was generated (see section 3.1.4) with the common name equal to the xmpp-domain name and this together with the corresponding private key were stored on each of the chat servers. In addition the root certificate of the Certicate Authority was stored in the xmpp servers trust store.

One of the limitations of the XOChat client (see section 2.8) is that all contacts for a user must be predefined. In order for two users in different security domains to chat with each other they must both be manually registered in the other users' roster (also known as address list). In addition they must be configured to subscribe to messages from each other.

Finally, in order for the chat servers to be able to connect to the Chat Guard, they must resolve the domain name of the remote server to the IP address of the guard protocol handler. Since no DNS was available within the testbed, the hostfile of the hosting machine was modified.

[3] https://www.igniterealtime.org/projects/openfire/

3.1.3 Clients

Chat clients based on the XMPP platform are commonly available, both as opens source software and commercial software. In the testbed we used two different XMPP clients, Spark[4] from Igniterealtime and the XOChat prototype developed by Thales Norway AS. The latter was the only available client that included functionality for both labelling and signing messages. The Chat Guard uses labels and signatures included in messages when performing release control. Labeling and signing is thus a requirement when chatting with users in another domain through the Chat Guard. The Spark clients were used for general XMPP testing.

Since no DNS was available the hostfile of the clients was also modified to include IP address information to the chat servers. Manipulation of the host file was chosen since no DNS was available.

3.1.4 Certificates

For the testbed, certificates from a Thales Certificate Authority (CA) were used. For simplicity both security domains used the same CA, and live certification and CRL checking were disabled. The CA root certificate and CRLs were pre-distributed to the Chat Guard, both XMPP servers and chat clients.

3.2 Testing with the SMART initiative

Each year the Norwegian Armed Forces execute a number Concept Development and Experimentation (CD&E) activities. In 2016, the activity "EP 1667 SMART – Pervasive common situational awareness at the individual soldier level" investigated whether commercial smart technology, such as smart phones and tablets, could be used to provide situational awareness to units with little or no equipment available today [12]. The activity was executed by the Norwegian Defence Research Establishment (FFI) and sponsored by the Norwegian Home Guard. All systems developed and used during the CD&E activity were in the unclassified domain. At the same time security mechanisms were implemented to protect the information from public disclosure and the end goal is an unclassified but secure system.

The SMART activity used chat as an integrated part of the experiment. It was thus natural to see the Chat Guard activity in connection with this since drawing experience from testing the guard in a close to real life operational scenario could be very valuable. Also, important lessons learned from operational usage of chat can be drawn from the SMART activity. This includes how chat is used in an operational setting, what functionality is necessary, and what is nice to have and what functionality the user can do without.

During the operational testing chat could be used for individual one-to-one sessions between soldiers or in pre-configured group chat sessions. Group chat constituted the bulk of chat

[4] https://www.igniterealtime.org/projects/spark/index.jsp

communication and it was used mainly to coordinate activities and share observation (including pictures). All chat users were part of the same domain, i.e., a single XMPP server were used for all users.

Based on the experiments from testing operational use of chat in the SMART activity, three major observations and lessons learned are relevant to the Chat Guard development. First, the use of chat rooms was extensive. The users coordinated and shared observations with each other in predefined groups rather than in one-to-one chat. Second, pictures of observations were shared using chat. Pictures were used for informing other users and as a catalyst for gathering more information on the observation. Finally, the users also wanted to know when a message was read and by whom, a type of read confirmation. The prototype Chat Guard did not support any of these and how this can be supported is described in section 4.

The Chat Guard was tested together with SMART at the end of the CD&E activity. In this demonstration chat between a simulated higher echelon in a classified security domain and the unclassified SMART systems and users were enabled by adding the Chat Guard between the systems. The Chat Guard was configured to allow all messages from the low (SMART system) to the high security domain, and to allow unclassified and signed messages the other way. This allowed the use of the commercially available chat clients already used in the SMART activity without modification. At the same time the guard protected the high domain from leaking information. Only minor changes were done to the SMART system itself, including the configuration of certificates for server-to-server communication with the guard and the other XMPP server. The generic testbed setup presented in the previous section was re-used as far as possible.

4 Discussions

The overall aim of the Chat Guard is to enable users in different security domains to chat with each other in a secure way, while also preventing information leakage. When introducing a security mechanism there is often a trade-off between security and usability. This is also the case for the Chat Guard and this chapter elaborates on some of the choices made, what the effects are and if there are any mitigating measures that can be taken. This section also discusses other subjects identified during testing and verification, including subjects identified through the use of chat in the SMART activity.

For cost and interoperability reasons it is desirable to be able to use COTS products as far as possible within the “chat domains” (i.e. COTS chat clients and chat servers), and to use the Chat Guard and possibly other security components to provide the required filtering. The chat domain may include security features that could supplement or partly replace mechanisms within the

Chat Guard. This is reflected in the discussions below by noting that several functions must be configurable.

4.1 Address exposure

The Chat Guard prototype allowed XMPP addresses to pass unfiltered. Filtering may need to be applied in order to reduce or eliminate possibilities for information leakage.

The XMPP address format is defined in RFC 7622 [5] as follows:

[localpart "@"] domainpart ["/" resourcepart]

Of these, the "localpart" (also known as the "username") identifies an XMPP entity (i.e., user or chat room) within the context of a "domainpart". The "domainpart" typically identifies an XMPP server (e.g. using an IP address or domain name). "resourcepart" may identify a specific channel towards the user (e.g., a device), or provide a nick-name for a specific participant in a chat room. The "resourcepart" could be seen as a potential information leakage channel, and may therefore need special handling as described below.

XMPP addresses (including the "resourcepart") may be regarded as sensitive, since such addresses may possibly convey information on the command structure of the high side (assuming the "localpart" refers to actual users and "domainpart" refers to actual servers), as well as information on the operational capacity (total number of addresses and number of addresses used at a given time). This issue is similar to the Presence stanzas discussed in section 4.2.1, but addresses are also used in other stanzas (including Message stanzas).

The main use of addresses is obviously to identify the Chat users. This implies that if addresses are to be modified in any way, the modification must be reversible. One possible mechanism could be to convert the "localpart" into an anonymous form (e.g., including a random number), and saving the correspondence in a mapping table. In order to start a chat session, there must be some mechanism to connect to specific users, possibly using Presence stanzas where the addresses have likewise been made anonymous. The consequence of such a strategy is that Chat users on one side of the Guard cannot identify users on the other side by name. In an organizational chat service this need not be a problem, but for person-to-person chat this is likely to be unacceptable.

The "domainpart" should be subject to filtering, by only allowing pre-configured values (i.e., only allow communication with specific XMPP servers). The prototype only allowed one XMPP server on each side, but in a more realistic scenario there may be a need to allow more than one XMPP server on each side. In this case it is likely that there should be restrictions on which XMPP servers are allowed to communicate.

The "resourcepart" is typically used for providing a nick-name for participants in a chat room, and for identifying a specific device. None of these are required, with the exception of sending

messages directly to a specific chat room participant, and the Chat Guard may therefore discard this information.

A possible solution could be to use a list of valid “localpart” entries on the “high” side, with those entries not tied directly to individual users. The entries could then be said to be “roles”, and more than one user may be allowed to log on to a specific “role” at different times. This would create a scenario somewhere between organizational and personal chat.

One possible strategy for protecting addresses is to use a chat room (e.g., all users on the “high” side are seen only as part of a “high side chat room”). See handling of chat rooms in section 4.3.

Depending on the sensitivity placed on the address information, the solution may range from full exposure of all addresses, via “role” addresses and anonymous addresses, to “high” side chat rooms. The Chat Guard will have to allow all strategies, with possibility to use different strategies in the two directions.

4.2 Presence and IQ stanzas

A Chat user’s session starts with establishing a stream between the client and home server. If the user sends chat messages to a user at another Chat server, a server-to-server Stream between the two servers is required, and a new Stream is established automatically if no suitable stream is available. All stanzas are exchanged over the established streams.

The Chat Guard acts as a proxy for the server on the “other” side of the Chat Guard. Streams are therefore negotiated on both sides of the Guard, and no information pertaining to streams are passed to the “other” side. No stream can be established through the Guard. Stream errors are thus not expected to arrive at the Guard Core at all. If any do arrive, they should be dropped, after logging an Alarm Event to the fact in the Guard’s Alarm Log.

4.2.1 Presence stanzas

The Presence stanza provides other Chat users with information on who is logged on and thus available for chat sessions. The Presence stanza may also reveal location information, as the “resource” part of the JID may be used to differentiate between a user’s chat clients at different locations. This information may be sensitive, as it exposes the XMPP identities of the users to other users across the Guard, as well as the times they are on duty. Discarding Presence stanzas (as done by the Chat Guard prototype) eliminates this information leakage, but this makes the Chat service more difficult to use (out-of-band signaling, e.g. via messages or phone, might be needed to arrange a Chat session with the desired participants).

A better strategy would be to configure which users (addresses) are allowed to publish their presence. This could use a list of allowed real user addresses (on the low side), or a list of generic (anonymous) user addresses (on the high side). As an example, the high side could be allowed to publish “subject matter A online”, without publishing details on who or how many

are actually logged on. A technical implementation by the Chat Guard could be to have a configurable list of allowed users with a corresponding “anonymous” name, and sending a Presence stanza for this name when a Presence stanza for the first of those users is received. See also section 4.3 on Chat rooms below. Such a strategy could be extended to allow “any” users on the low side, thus filtering only the high-to-low Presence stanzas.

4.2.2 IQ stanzas

IQ stanzas are client-client (via one or two servers), client-server, or server-server. The Chat Guard prototype did not allow any IQ stanzas to pass through the Guard, for instance preventing the chat room functionality. In the final Chat Guard IQ stanzas will have to be handled in a more refined way.

Server-server IQ stanzas are exchanged between peers. The Guard negotiates these in each Protocol Adapter module, and never involves the Guard Core. Client-server and client-client IQ may pass through the Guard. These are used, among others, to query roster and chat room configurations, and to exchange status and capabilities information between clients.

IQ stanzas are not strictly necessary for XMPP messaging through the Guard to work, except that “discovery” queries must be supported for multi-user chat (chat room) to work across the Guard. Since both the “get” and “result” stanzas may contain quite a bit of information, a Guard that supports multi-user chat should be able to define filtering rules for this group of IQ stanzas.

The table below summarizes the roster and discovery IQ queries that should be allowed through the Guard, possibly after filtering and identity transformations):

IQ type	**IQ namespace**	**Request param**	**Result type**
get	jabber:iq:roster		
result	jabber:iq:roster		<item/> [1..n]
get	http://jabber.org/protocol/disco#items		
result	http://jabber.org/protocol/disco#items		<item/> [1..n]
get	http://jabber.org/protocol/disco#info	<identity/>	
result	http://jabber.org/protocol/disco#info		<identity/> [1..n]

A roster query result will return an item-not-found error if no roster is defined for the user. If the roster is defined, but empty, the return stanza will contain a query element with no contained <item/> elements. A disco#items may return service-unavailable, forbidden or not-allowed errors.

Apart from the central IQ query namespaces discussed above, there are a number of more or less specialized IQ namespaces that are less central from the Guard perspective:

Namespace	Purpose	Comment
Jabber:iq:auth	Obsolete (defined non-SASL authentication)	Dropped by the Chat Guard
Jabber:iq:gateway	Client to gateway/proxy for legacy IM	Dropped by the Chat Guard
Jabber:iq:last	Query entity's last activity	IQ:last may reveal sensitive information, so it should be possible to configure a drop for this.
Jabber:iq:oob	Out of Band Data URIs	IQ:oob may be useful, but the current Guard is not technically ready to support it. If support is implemented, IQ:oob support should be supplemented with Guard Filtering Rules.
Jabber:iq:privacy	Core functionality for setting privacy (e.g. who may get presence?)	Drop. IQ:privacy would normally not be seen, as this is between a client and the server it is logged in to. If this arrives, it would imply a login attempt through the Chat Guard, which is not supported.
Jabber:iq:private	Store arbitrary XML on server	Dropped by the Chat Guard
Jabber:iq:register	E.g. dynamically register with a server	Dropped by the Chat Guard
Jabber:iq:roster	Retrieve user's roster from server	Dropped by the Chat Guard
Jabber:iq:rpc	Perform XML-RPC call	Dropped by the Chat Guard
Jabber:iq:search	Non-standard (search information repositories)	Dropped by the Chat Guard
Jabber:iq:version	Return software version representing the XMPP entity	IQ:version is probably never very sensitive, so this may be always-allow. If the version number should be hidden to prevent easy finger-printing, this IQ may be dropped.

4.2.3 Stanza errors

The Chat Guard prototype accepted responsibility for a stanza on reception, rather than when the stanza was delivered to the XMPP server on the other side. This could cause stanzas to be lost. A full implementation would need to either act as a store-and-forward unit (i.e. store any received stanzas until they can be delivered to the next unit), or return error indications in case of problems. The Mail Guard is based on the latter principle, and the Chat Guard should therefore also use this strategy. Some stanza errors may thus be generated by the Chat Guard. This is noted in the table below.

The stanza errors listed below are defined in the standard. Most of these may pose information leak potential, so in scenarios involving at least one security domain with sensitive information, it may be desirable to configure which error types to permit through the guard. Configurations should be separate for each direction, and logically belongs within the “directional” section of the Configuration Vector.

No XMPP functionality unconditionally relies on any of these error types to pass freely, so dropping all error messages is functionally safe (and explicitly permitted by the standard’s Security Considerations).

Error name	**RFC explanation**	**Comment**
Bad-request	The sender has sent XML that is malformed or that cannot be processed (e.g., an IQ stanza that includes an unrecognized value of the 'type' attribute); the associated error type SHOULD be "modify".	If received, this is the Guard’s responsibility, and it shall not be passed through.
conflict	Access cannot be granted because an existing resource or session exists with the same name or address; the associated error type SHOULD be "cancel".	Configurable error type. If revealing resources/sessions is undesirable, drop this.
feature-not-implemented	The feature requested is not implemented by the recipient or server and therefore cannot be processed; the associated error type SHOULD be "cancel".	Configurable error type. If revealing features is undesirable, drop this.
forbidden	The requesting entity does not possess the required permissions to perform the action; the associated error type SHOULD be "auth".	If received, these are targeted at the guard, and will never be passed through.
gone	The recipient or server can no longer be contacted at this address (the error stanza MAY contain a new address in the XML character data of the <gone/> element); the associated error type SHOULD be "modify".	Configurable error type. If revealing individual status is undesirable, drop this.

internal-server-error	The server could not process the stanza because of a misconfiguration or an otherwise-undefined internal server error; the associated error type SHOULD be "wait"	Configurable error type. This indicates a temporary error, and most likely contains no sensitive information, so it may be passed through.
item-not-found	The addressed JID or item requested cannot be found; the associated error type SHOULD be "cancel".	Configurable error type. If revealing individual status is undesirable, drop this. This could be used if a stanza cannot be delivered to the destination XMPP server.
jid-malformed	The sending entity has provided or communicated an XMPP address (e.g., a value of the 'to' attribute) or aspect thereof (e.g., a resource identifier) that does not adhere to the syntax defined in [3]; the associated error type SHOULD be "modify".	If this is received, it’s related to the Guards reconstructed stanza, and should be handled by the Guard, so the message should not be passed through.
not-acceptable	The recipient or server understands the request but is refusing to process it because it does not meet criteria defined by the recipient or server (e.g., a local policy regarding acceptable words in messages); the associated error type SHOULD be "modify".	Configurable error type. Local policy decisions is a core Guard function, which often is not communicated back to the sender, but instead registered in an audit trail for Manager consideration. In this case, a modify response may be seen as too revealing. Option to drop, or to return a not-allowed instead.
not-allowed	The recipient or server does not allow any entity to perform the action; the associated error type SHOULD be "cancel".	Configurable error type. Also core Guard decisions, which we possibly want to restrict to the audit trail. Option to drop.
not-authorized	The sender must provide proper credentials before being allowed to perform the action, or has provided improper credentials; the associated error type SHOULD be "auth".	If received, these are targeted at the Guard, and will never be passed through.

payment-required	The requesting entity is not authorized to access the requested service because payment is required; the associated error type SHOULD be "auth".	Configurable error type. May be used to deduce restricted information from the opposite side. If payment-required services are allowed in a secure scenario at all, this may optionally be dropped, to reduce the leak potential.
recipient-unavailable	The intended recipient is temporarily unavailable; the associated error type SHOULD be "wait" (note: an application MUST NOT return this error if doing so would provide information about the intended recipient's network availability to an entity that is not authorized to know such information).	Configurable error type. May reveal information regarding individual presence, but is probably often a desirable function. Drop if considered sensitive. This could be used if a stanza cannot be delivered to the destination XMPP server.
redirect	The recipient or server is redirecting requests for this information to another entity, usually temporarily (the error stanza SHOULD contain the alternate address, which MUST be a valid JID, in the XML character data of the <redirect/> element); the associated error type SHOULD be "modify".	Configurable error type. May reveal information regarding individual presence, but is probably often a desirable function. If such “readdressing” is considered sensitive, this should be dropped (possibly one-way, if the information is not sensitive in the other domain)
registration-required	The requesting entity is not authorized to access the requested service because registration is required; the associated error type SHOULD be "auth".	Configurable error type. May be used to deduce restricted information from the opposite side. If registration-required services are allowed in a secure scenario at all, this may optionally be dropped, to reduce the leak potential.

remote-server-not-found	A remote server or service specified as part or all of the JID of the intended recipient does not exist; the associated error type SHOULD be "cancel".	Configurable error type. No sensitive information leak (apart from possibly to map available remote servers/services). Optionally drop. This could be used if a stanza cannot be delivered to the destination XMPP server.
remote-server-timeout	A remote server or service specified as part or all of the JID of the intended recipient (or required to fulfill a request) could not be contacted within a reasonable amount of time; the associated error type SHOULD be "wait".	Configurable error type. This is a temporary problem: No sensitive information leak (apart from possibly to map available remote services). Optionally drop.
resource-constraint	The server or recipient lacks the system resources necessary to service the request; the associated error type SHOULD be "wait".	Configurable error type. This is a temporary problem: No sensitive information leak, but may conceivably be used in preparation for, or in conjunction with a (D)DOS attack. Optionally drop.
service-unavailable	The server or recipient does not currently provide the requested service; the associated error type SHOULD be "cancel".	Configurable error type. May pass information about available services through the Guard. Optionally drop. This could be used if a stanza cannot be delivered to the destination XMPP server.
subscription-required	The requesting entity is not authorized to access the requested service because a subscription is required; the associated error type SHOULD be "auth".	Configurable error type. May be used to deduce restricted information from the opposite side (e.g. the opposite side has an indicated entity, but that entity has not granted the requestor the right to subscribe to its status information). Optionally drop.

undefined-condition	The error condition is not one of those defined by the other conditions in this list; any error type may be associated with this condition, and it SHOULD be used only in conjunction with an application-specific condition.	Unknowns of any kind provide possible security bypass mechanisms, so this should be dropped.
unexpected-request	The recipient or server understood the request but was not expecting it at this time (e.g., the request was out of order); the associated error type SHOULD be "wait".	This is in effect an “unknown”, which in a secure setting normally is not wanted, so this error should be dropped.

4.3 Multi-User Chat (Chat rooms)

The prototype Chat Guard did not support Multi-User Chat (MUC), as a result of the choice to not support IQ or Presence stanzas, and partly due to the stanza re-signing issue described below. Group chat is however a very effective and much used way of communicating with more than one person. In fact, experience from the SMART activity shows that group messaging and discussions are used far more than one-to-one chat, especially when coordinating actions and sharing observations. Adding support for group chat is thus important.

The primary challenge with multi-user chat is the stanza signing. The multi-user chat protocol specifies that “groupchat” messages arriving at the chat room shall be duplicated into separate “chat” messages to each chat room occupant, rewriting the from/to attributes of the message. This invalidates the signature of the message.

Therefore, for multi-user chat to work, it seems necessary to employ an XMPP server which is able to verify the original signature, and resign each of the duplicates. Once such a server is employed, the Guard only needs to be extended to allow “groupchat” message types.

As discussed in chapter 4.2.2, in order to support chat rooms across the Guard, Discovery as specified by *XEP-0030: Service Discovery*[5] must be supported.

[5] https://xmpp.org/extensions/xep-0030.html

Multi-user chat discovery is typically performed like this

1. The client sends an initial disco#items get query to the XMPP server the client wants to find chat rooms on.

2. The server returns a disco#items result IQ stanza containing a list of registered items (some of which may be chat rooms)

3. The client sends a disco#info get query to the server for each of the items in the list returned in the previous step.

4. The server returns a disco#info query result IQ stanza containing an <identity/> element and a list of <feature/> elements for each item indicated in the previous step.

5. The client checks if the <identity> contains the property category=”conference”, and if one of the <feature/> elements contains a var=”http://jabber.org/protocol/muc” IQ namespace reference. If so, the JID of the item in question identifies a chat room service.

Discovering the actual chat rooms of the service is done in the same way, a “disco#items” sent to the JID of the chat room services will provide a list of the chat rooms hosted by the service, and a “disco#info” directed to the JID of a particular chat room item will provide a list of features configured for the chat room.

Beyond message stanzas, multi-user chat relies heavily on presence stanzas, so for multi-user chat support, the Guard must accept Presence stanzas, and provide for defining presence-related filtering rules in the guard’s Configuration Vector.

4.4 Attachments and content checking

The experiments performed as part of the SMART activity indicates that there may be strong operational requirements for being able to include pictures as attachments in chat messages, thereby imposing a requirement on the Guard to support/allow such attachments. However, in the current prototype implementation of the Guard attachments are not allowed.

Attachments represent additional security risk, as they may leak sensitive content or contain malicious content such as malware. If these risks are deemed acceptable, the Guard could simply be configured to allow specific attachment types to be included. In that case, no security check would be performed on the attachment apart from verifying that the security label of the message is releasable according to policy and verifying that the signature is correct and covers the attachments. However, some type of content checking will likely be required in many scenarios.

ICAP [13] compliant plug-in content checkers are supported through the content checking interface of the Guard. This allows including both generic content checkers, such as antivirus, as well as content checkers for specific attachment types. The selection of which attachment checker(s) to invoke may potentially be performed by an orchestrating content checker, thereby not requiring this functionality to be provided by the Guard itself.

For text content, it is quite common to use a dirty-word checker, scanning for specific words considered to indicate sensitive content. For attachments, this requires the content checker to be able to read the specific type of attachment (e.g., file type). It is also possible to use more advanced data loss prevention (DLP) solutions for content checking to detect sensitive content, e.g., based on techniques from machine learning or information retrieval [14]. While known sensitive documents/content can be detected with high accuracy using such methods, it is more difficult to detect transformed data leaks (i.e., where known sensitive information has been modified/rewritten and/or mixed with other text) or previously unknown sensitive content. To avoid a prohibitively high number of false alarms/positives, detection may be performed for each entity (e.g., transmitting user) on a longer timescale (i.e., over multiple messages), detecting more long-term discrepancies between the security labels applied by a user and those determined by the DLP solution [15].

Apart from antivirus and image metadata, content checkers for images may use fingerprinting techniques to check whether an image is closely related to a known sensitive image. Alternatively, image recognition/classification may be used to classify an image into one or more categories, where some categories may be allowed for release or not. By performing character recognition, images could potentially also be scanned for sensitive text content.

Transcoding could potentially be applied to protect against malicious content being transferred through media attachments, however, it should be noted that this would change the attachment (breaking the signature) and is not a content checker as such.

4.5 Read confirmation

One of the key features of chat is presence, which lets you know that the people you are exchanging messages with actually are available. As described in section 4.2.1, presence messages were not allowed to pass through the prototype Chat Guard. The effect is that users on the different sides must send messages in the blind, not knowing if the recipient is online, away or otherwise not able to reply.

Adding a form of read confirmation may be one way of reducing the effect of the missing presence information. With read confirmation a message is sent back to the sender when the message is read by the receiving user. This could be solved by establishing a manual user procedure where the recipient actually types “message read” in the chat stream similar to how two users interacts on voice communication. Using such manual procedures is however error prone and an automatic solution should be implemented.

Implementing read confirmation requires establishing a protocol between sender and receiver, either standardized or proprietary. Either way will require implementing support for these standards within the chat clients. Using standardized specifications is recommended and within the XMPP community several specifications exist that either tries to solve this or partially do. The "XEP-0333: Chat markers"[6] describes a solution for marking the last received, displayed and acknowledged message in a chat. This specification is however still considered experimental and not recommended for production systems until it reaches draft status. At the time of writing the specification has not been updated since October 2015.

Using the specification "XEP-0184 Message delivery receipts"[7] a sender can request notification when the message is sent to a client under the control of the recipient. This does however not imply that the message is actually read by the recipient. Some indication of what the recipient is doing can also be provided by the specification "XEP-0085 Chat state notification"[8] which is used to indicate whether a chat partner is typing, paused, inactive or gone. This information can at least provide some indication on whether the message has been read or not.

Read confirmation is also a useful functionality even though presence information is available. In many cases it very useful for the sender to actually know if the message has been read. It was also one of the features requested by the users of the SMART system.

4.6 Border protection devices

The Chat Guard constitutes the mechanism that both connects and keeps the two security domains separate by providing the functionality described in section 2.1. Depending on the scenario, other border protection devices may also be necessary. The Chat Guard will typically be deployed as part of a larger cross-domain sharing solution, for instance a NATO Information Exchange Gateway (IEG) or a Norwegian SIU ("Sikker Informasjons-Utveksling").

Other protection mechanisms serve two purposes, first adding to the total protection of the internal domain and second to protect the Chat Guard itself. Border protection devices may include firewalls, virus and malware checking, content checkers, intrusion detection systems (IDS), de-militarized zones (DMZ) and other cyber threat detection systems. Which border protection system that is required depends on the scenario and the risk involved.

[6] https://xmpp.org/extensions/xep-0333.html
[7] https://xmpp.org/extensions/xep-0184.html
[8] https://xmpp.org/extensions/xep-0085.html

4.7 Handling of certificates

Securing XMPP streams is done with SASL/TLS authentication and encryption. This implies use of certificates and cryptographic keys. In the prototype, certificates issued by a self-signed Thales CA were used, with corresponding private keys stored as plain files.

The certificate of an XMPP server is identified by having the “xmpp-domain” name as Common name. In the prototype, both the real XMPP server, and the Guard proxying interface in the other domain used the same certificate and private key.

In a real scenario, one would wish to use separate certificates for the Guard proxying interface and the real XMPP server. The two exist in separate security domains, which would normally be supplied with separate certificate authorities and lookup services. This, and the fact that the certificate lookup is based simply on the CommonName attribute of the certificate, provides a solution where there are different valid certificates for the XMPP server in the two security domains. In this scenario, there is no need for the Guard and the real XMPP server to use the same certificates and keys. In a real scenario, it would also be expected that all private key handling is performed via a Hardware Security Module (HSM).

The Chat Guard will normally require that XMPP stanzas are digitally signed in order to ensure that release criteria (e.g. security label) are correctly bound to the content by an authenticated user. This is important on the “high” side to ensure only approved information can be released, but might also be important on the “low” side as a mechanism to verify the integrity of any data delivered to the “high” side. In some scenarios this may not be possible, e.g. because there is no PKI on the “low” side. The Guard must allow such configurations.

Certificates and CRLs are assumed to be issued by separate PKIs on the two sides, and the Guard therefore has separate interfaces for accessing certificates and CRLs on each side. This functionality must allow different strategies for CRL distribution point, CRL caching, and possibly configurable certificate policies.

4.8 Chat clients

The usage scenario with a number of chat clients logging on to an XMPP server introduces the question of how chat clients can be authenticated by the Chat Guard. Assuming the chat client can digitally sign the information using a key referenced to a certificate, the signature may be used as authentication. If the XMPP server generates the signature (or if signatures are not used), the Chat Guard must rely on the XMPP server to perform authentication of the chat client. This may have a bearing on which chat clients and XMPP servers that can be used in a given scenario, particularly on the “high” side.

The Chat Guard requires that stanzas can be validated in order to release them to the other side. This validation is typically based on confidentiality labels and digital signatures, but other strategies are possible as shown below.

Normally, a Chat client must include a confidentiality label on each Message stanza, and bind the security label to the content using a digital signature. This is not a standard feature in COTS Chat clients, thus requiring either a purpose-built Chat client or extension of an existing Chat client. A prototype Chat client was built as part of this activity, allowing Message stanzas to be labeled and signed (see section 2.8). It also supports verification of signature and label on incoming stanzas. Beyond this, only a bare minimum of XMPP functionality was included.

There may be cases where Chat users are physically connected to the "high" side, but the users themselves are trusted to only include "approved" information within Message stanzas. In such cases it might be acceptable to allow the XMPP server to perform labeling and signing on behalf of the clients, thus allowing the use of standard Chat clients.

There may also be cases where labeling and signing can be dispensed with altogether (in effect stating that "this XMPP server is trusted to only send releasable information to the Guard"). In such cases it clearly becomes vital to ensure the stanzas actually come from the correct XMPP server.

The Chat Guard should allow the different strategies above, and should allow the strategy to be different in each direction.

However, there are at this time no other known, available Chat Clients that support this, so for a full Chat Guard solution, it may be necessary to extend the prototype Chat Client towards full XMPP support.

4.9 Multiple XMPP servers in each security domain

The Chat Guard prototype supported only one XMPP server in each of the attached security domains. This was chosen as a means to reduce the complexity of the prototype, while still being able to demonstrate the central principles.

The XMPP protocol is designed for direct communication between XMPP servers, i.e. messages are never relayed via a third server. Figure 4.1 shows a simple typical scenario with four separate XMPP servers. The servers communicate directly with each other and messages are never relayed by intermediate XMPP servers.

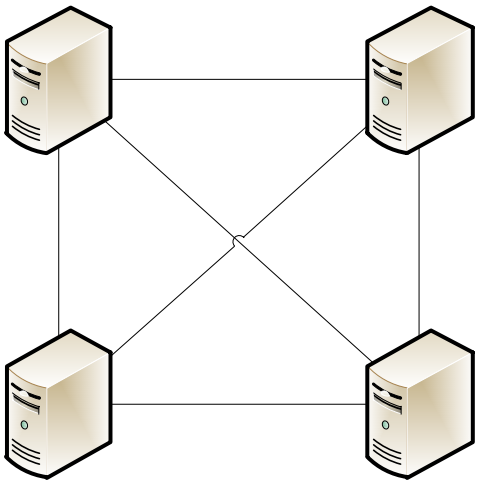


Figure 4.1 Multiple XMPP servers

If we consider the servers to actually belong in two separate security domains which we want to separate by one or more security gateways, the XMPP network might look like Figure 4.2.

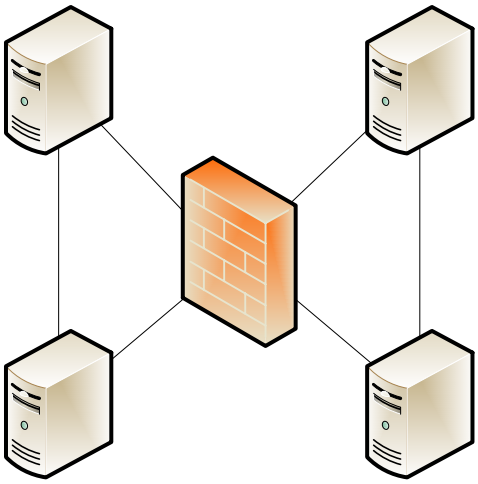


Figure 4.2 Multiple XMPP servers in separate domains

XMPP stanzas are sent to servers based on the domain part of the recipient XMPP address, typically using DNS to look up the IP address from a fully qualified hostname. The Chat Guard may need to filter the address information to avoid leakage as described in chapter 4.1, but this does not in itself prevent the Chat Guard from connecting to multiple XMPP servers on one or both sides. But it does require any address mapping to be reversible, at least for the "domainpart" component.

Note, however, that this assumes all XMPP servers on each side use the same PKI when signing messages. Having separate PKIs for each XMPP server could result in significantly more complexity related to certificate and CRL handling. Also, more advanced filtering may be required if there are restrictions on the connectivity between XMPP servers (e.g. server “A” on the “high” side is only allowed to communicate with servers “X” and “Y” on the “low” side).

As seen from the Guard, there are two main differences from the prototype:

- The XMPP Proxy listener must be able to accept and support multiple, concurrent XMPP streams, and to masquerade as a number of different XMPP servers in each security domain. This also implies that DNS queries regarding XMPP servers across the Chat Guard must resolve to the IP address on the “near” side of the Guard.

- XMPP stanzas arriving from the different XMPP servers will have different JID “domainparts”. The Guard filtering rule engine is designed to support this (a JID, after all, is a form of originator/recipient address). Apart from this, the Guard must implement a simple “routing” mechanism, to ensure that incoming stanzas are routed into the correct outgoing XMPP stream.

4.10 Consequences of proxying

Since the Guard acts as a transparent proxy, a connecting XMPP server will believe that it has an established stream towards the destination XMPP server, while it is fully possible that the destination is not available. This may be discovered at a later stage, when the Guard releases the stanza to the recipient, and the attempt to establish the XMPP channel fails. In such a scenario, the prototype Guard would silently drop the stanza without notifying the sender.

A better solution would be to allow the system manager to configure whether this should be indicated to the sender in the form of an appropriate error return.

5 Conclusions

Informal messaging is rapidly becoming an important means of communication, both one to one and in groups. However, due to differences in classification, users of military systems are often prevented from taking advantage of this opportunity. In this document we have described a prototype guard solution that enables users in different security domains to send chat messages to each other.

The development and testing of the prototype Chat Guard has proven that is both feasible and useful to implement a high assurance guard for chat messaging. Re-using and extending an already existing guard platform has enabled us to create a working prototype in a relatively short amount of time. Reuse of an already existing high assurance guard platform should also simplify the process of certification since much of the security critical code is unchanged.

When creating security mechanisms it is critical to find the right balance between protection and functionality. Information may be compromised if too lax and the system may be rendered useless if too strict. Which information and types of messages that should be allowed is dependent on the scenario and the finished Chat Guard should be adaptable by configuration. Also the risk involved and the level of risk that is acceptable will play an important role when configuring and deploying the guard. There is often an asymmetry to what is allowed and not, usually more elements are allowed to flow from low to high, than the other way. This fact may be utilized when creating services using the Chat Guard.

The need for solutions that enables users to work together across systems of different classifications is well-known. The Chat Guard prototype has shown that it is possible to enable this for chat while still providing the needed level of security. When this technology can be used is dependent on the scenario and the risk involved and the level of risk that can be accepted.

The testing of the Chat Guard prototype has identified some areas that may need further work. Chat rooms have been requested as a highly desired service. Also, there is an operational need to see that chat messages have been read. Finally, there is a need to provide Chat clients (or in some scenarios XMPP servers) that can add security labels and digital signatures, at least on the high side.

References

[1] R. Haakseth, N. A. Nordbotten, B. Kristiansen, Ø. Jonsson and M. Andreassen, “CD&E EP 1328 Guard for cross-domain information exchange”, FFI-rapport 2014/01182, 2014, (Begrenset)

[2] R. Haakseth, N. A. Nordbotten, B. Kristiansen and Ø. Jonsson, “A high assurance guard for use in service-oriented architectures”, IEEE International Conference on Military Communication and Information Systems, 2015

[3] P. Saint-Andre, ”Extensible Messaging and Presence Protocol (XMPP): Core”, Internet Engineering Task Force (IETF) RFC 6120, 2011

[4] P. Saint-Andre, ”Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence”, Internet Engineering Task Force (IETF) RFC 6121, 2011

[5] P. Saint-Andre, ”Extensible Messaging and Presence Protocol (XMPP): Address Format”, Internet Engineering Task Force (IETF) RFC 7622, 2015

[6] NATO Standardization Organisation (NSO), “Confidentiality Metadata Label Syntax”, ADatP-4774, 2016

[7] NATO Standardization Organisation (NSO), “Metadata Binding Mechanism (DRAFT), ADatP-4778, 2016

[8] D. Box et.al., “Web Services Addressing (WS-Addressing)”, W3C recommendation, 2004

[9] K. Wrona and N. Menz, “Protection profile for the NATO high assurance abac guard (haag) version 1.3”, NCIA Technical Report TR-2012-SPW0084-18-13-4, NATO Communications and Information Agency (NCIA), 2013.

[10] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1”, Internet Engineering Task Force (IETF) RFC 4346, 2006

[11] A. Melnikov and K. Zeilenga, “Simple Authentication and Security Layer (SASL)”, Internet Engineering Task Force (IETF) RFC 4422, 2006

[12] F. T. Johnsen, M. R. Brannsten, A-K. Elstad, T. H. Bloebaum and F. Mancini, “SMART: Situational awareness experiments with the Norwegian home guard using Android”, FFI-rapport 2017/0073

[13] J. Elson and A. Cerpa, “Internet Content Adaption Protocol (ICAP)”, Internet Engineering Task Force (IETF) RFC 3507, 2003

[14] K. W. Kongsgård, N. A. Nordbotten, F. Mancini and P. E. Engelstad, “Data Loss Prevention Based on Text Classification in Controlled Environments”, Information Systems Security, Springer, 2016.

[15] K. W. Kongsgård, N. A. Nordbotten, F. Mancini and P. E. Engelstad, “An Internal/Insider Threat Score for Data Loss Prevention and Detection”, Proc. ACM International Workshop on Security and Privacy Analytics, 2017.

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

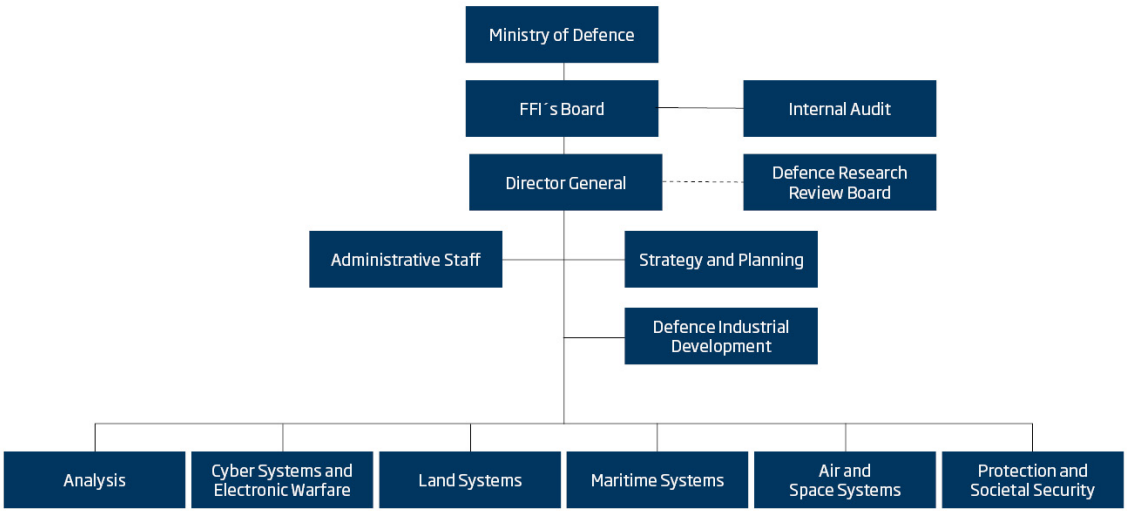
FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFI's organisation



Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment