



IceWall SSO

Version 10.0

IceWall Cert Protocol 2.0 Developer's Manual

August 2010

Printed in Japan



HP Part No. B1544-97013

Rev.111012A

Notice

The information contained in this document is subject to change without notice.

Meticulous care has been taken in the preparation of this document, however, if a questionable or erroneous item, or an omission of content is found, please contact us.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damage in connection with the furnishing, performance or use of this document.

The copyright of this document is assigned to Hewlett-Packard Development Company, L.P. No part of this document may be photocopied or reproduced without prior written consent by Hewlett-Packard.

This manual and media, such as the CD-ROM supplied as a part of the product's package, are only to be used with this product.

IceWall is a trademark of Hewlett-Packard.

Adobe is a trademark of Adobe Systems Incorporated in the United States and/or other countries.

Intel, the Intel logo, Intel. Leap ahead., Intel. Leap ahead. logo, Itanium and Itanium Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

– Table of Contents –

1	Introduction	1
1.1	Version designations in the text	1
2	IceWall Cert Protocol 2.0	2
2.1	What is IceWall Cert Protocol?	2
2.2	Message configuration	2
2.3	Restrictions	2
3	Details of IceWall Cert Protocol 2.0	4
3.1	List of Messages	4
3.2	Message format	5
3.2.1	Request message header	5
3.2.2	Response message header	6
3.2.3	Body	7
	Login request using user ID and password	10
	Forced login request using user ID and password	12
	Login request using client certificate and password	14
	Forced login request using client certificate and password	17
	Login response using user ID and password	19
	Response for login request using client certificate and password	22
	Access control request (user ID and password)	25
	Access control request (client certificate and password)	27
	Access control response (user ID and password)	30
	Access control response (client certificate and password)	32
	Password change request	34
	Password change response	36
	Logout request	38
	Logout response	40
4	Communication Message Encryption Library 10.0	42
4.1	What is communication message encryption library?	42
4.2	Extending the communication message encryption library	42
4.2.1	Communication message encryption library for product standard encryption	43
4.2.2	Communication message encryption library for custom encryption	43
4.3	Development of the communication message encryption library	44
4.3.1	Development procedure of the communication message encryption library for product standard encryption	45
4.3.2	Development procedure of the communication message encryption library for custom encryption	46
4.4	Notes on using the communication message encryption library for custom encryption	47
5	Specifications of the Communication Message Encryption Library for Product Standard Encryption	48
5.1	Callback functions	48
	IW_ExEncrypto	49
	IW_ExDecrypto	50

5.2 API function	51
IW_ExCPTCreateCrypto	52
IW_ExCPTReleaseCrypto.....	53
6 Library Specifications of the Communication Message Encryption Library for Custom Encryption 10.0	54
6.1 Callback functions	54
CS_ExEncrypto	55
CS_ExDecrypto	56
6.2 API function	57
IW_ExCPTCreateCrypto	58
IW_ExCPTReleaseCrypto.....	59
CS_ExGetIWEncryptedType	60
7 Notes and Remarks on Implementing ICP	61
7.1 Remarks	61
7.2 Restrictions	61
8 Notes and Restrictions on Developing the Communication Message Encryption Library.....	62
8.1 Remarks	62
8.2 Restrictions	62
9 ICP 2.0 Client Sample	64
9.1 Client sample 1	64
9.2 Client sample 2	65
9.3 Agent sample (for Apache HTTP Server 1.3.x)	66
9.4 Sample source (C language).....	68
9.4.1 Client sample 1 (sample1.c).....	68
9.4.2 Client sample 2 (sample2.c).....	71
9.5 Sample source (Java language).....	74
9.5.1 Client sample 1 (Sample1.java).....	74
9.5.2 Client sample 2 (sample2.java)	78
9.6 Sample source (Agent sample: mod_sample.c).....	84
10 Sample Communication Message Encryption Library.....	96
10.1 Sample source file (iwcrypto.c).....	96
10.2 Header file (iwcrypto.h).....	97
10.3 Makefile for the Forwarder (HP-UX edition: Makefile) 10.0	98
10.4 Makefile for the Forwarder (Linux edition: Makefile) 10.0	99
10.5 Makefile for the Authentication Module (HP-UX edition: Makefile).....	99
10.6 Makefile for the Authentication Module (Linux edition: Makefile).....	100

1 Introduction

This manual describes the message format and specifications of the module-to-module communication messages (IceWall Cert Protocol 2.0) implemented in the IceWall SSO Version 8.0 or later, and provides information necessary to develop libraries for encrypting or decrypting the messages.

By implementing the specifications described in this manual, you can create a custom module that communicates with the Authentication Module for authentication and authorization similar to the Forwarder or agent.



The contents of this manual are roughly divided into the following two categories:

- (1) IceWall Cert Protocol 2.0 message specifications
- (2) Reference for developing communication message encryption/decryption libraries for the Authentication Module

Thoroughly read these contents to make use of the potential authentication and authorization functionality provided by the IceWall SSO.

1.1 Version designations in the text

The table below gives the meanings of the version designations added to the text.

Designation	Meaning
	An item added to the version enclosed in the square. In this case, the designation indicates the item was added to 10.0.
	An item where the specification was changed or function added to the version enclosed in the oval. In this case, the designation indicates a specification change or function addition to 10.0.

2 IceWall Cert Protocol 2.0

This chapter and the following chapters describe the specifications of the IceWall Cert Protocol (hereafter referred to as "ICP") 2.0.

2.1 What is IceWall Cert Protocol?

IceWall Cert protocol is a protocol for message communications between the modules configuring the IceWall SSO system.

2.2 Message configuration

ICP 2.0 is mainly configured by the following two types of messages:

- (1) Request messages
- (2) Response messages

A request message is sent from a module (i.e., the client) to the Authentication Module. When the Authentication Module receives this message, it executes a process according to the received message.

A response message is sent back from the Authentication Module to the requesting source to indicate the processing result for the request from the client module. This message stores the request result.

In a single sequence, one response is returned to each request. Multiple responses are not returned to a single request.

2.3 Restrictions

The following restrictions are applicable to ICP 2.0.

- The message is handled as text data and cannot contain binary data.
- It is assumed that each message is encrypted before communication. The encryption and decryption algorithm used for the message must be identical to that of the "communication message encryption library" used by the Authentication Module.
- For encryption and decryption of messages, you can use either the standard communication message encryption library or the communication message encryption library for custom encryption.
- If the data length of the request message header (Content-length) differs from the data length of the body, the Authentication Module returns a system error as the response. In this case, the body of the response is encrypted by the

communication message encryption library for standard encryption. In addition, the following message is output to the error log of the Authentication Module: ⑩.⑩

Fatal: Content-length Error.[length set to Content-length] Body=[actual body length][EL60606-05013]

The following message is also output in version 8.0 R3 and later:

Fatal: EC16408-25115 Not Found Content-Length.
Fatal: EC16407-25114 Content-Length is 0.
Fatal: EC16409-05013 Content-length Error. [%d] Body=[%d]

- The supported character code set is Shift_JIS only.

3 Details of IceWall Cert Protocol 2.0

This chapter describes the details of ICP 2.0.

3.1 List of Messages

The following lists the messages implemented in ICP 2.0.

Message	Description
ReqLoginUID	Request for login using the user ID and password.
ReqForceLoginUID	Request for forced login using the user ID and password.
ResLoginUID	Response for a login or forced login request using the user ID and password.
ReqLoginCERT	Request for login using the client certificate and password.
ReqForceLoginCERT	Request for forced login using the client certificate and password.
ResLoginCERT	Response for a login or forced login request using the client certificate and password.
ReqAccessUID	Request for access control after login using the user ID and password.
ReqAccessCERT	Request for access control after login using the client certificate and password.
ResAccessUID	Response for an access control request after login using the user ID and password.
ResAccessCERT	Response for an access control request after login using the client certificate and password.
ReqPasswdChg	Request for password change.
ResPasswdChg	Response for a password change request.
ReqLogout	Request for logout.
ResLogout	Response for a logout request.

3.2 Message format

Both the request and response messages consist of a header and body.

3.2.1 Request message header

The header of a request message consists of the following data.

REQ / ICP/2.0	CRLF	Request line
X-iw-encrypted: [enc-type]	CRLF	Encryption method line
Content-length: [length]	CRLF	Data length line
. . . .		Arbitrary header line
CRLF		Header separator

Name	Mandatory	Description
Request line	○	Set the keyword "REQ" for request, fixed character "/", and protocol version. Separate the elements with a blank space. Specify the protocol version by separating the protocol name and version number with "/". The protocol version is fixed to "REQ / ICP/2.0" for ICP 2.0. If the request line is not defined correctly, it is treated as an invalid ICP 1.0 request.
Encryption method line 10.0	×	Set the communication message encryption library used to encrypt the message body. Use the following procedure. <ul style="list-style-type: none"> Use libCryptoExit.sl for standard encryption to encrypt/decrypt the body. Set X-iw-encrypted header to "icewall." or Do not set the X-iw-encrypted header. Use libExCryptoExit.sl for custom encryption to encrypt/decrypt the body. Set a value other than "icewall" in the X-iw-encrypted header to indicate custom encryption.
Data length line	○	Set the data length of the message body. The Authentication Module obtains the message body based on the data length specified here. If the data length is different from the body section length, the Authentication Module does not return a response and disconnects the connection.

Name	Mandatory	Description
Arbitrary header line ⑩.①	×	<p>Set an arbitrary header. The format is "header name: header value." (Example) Client-Id: 0001 A line feed code is required at the end of the line to separate the header lines. You can add any number of header lines. You cannot use the following header names because they are reserved words.</p> <div><p>Content-length X-iw-encrypted X-iw-failback</p></div>
Header separator	○	<p>Enter a line feed code as the header separator. If you do not enter this header separator, the body is recognized as a part of the header and the entire message is treated as an invalid message.</p>

3.2.2 Response message header

The header of the response message consists of the following data.

ICP/2.0 [status] [msg]	CRLF	Response line
X-iw-encrypted: [enc-type]	CRLF	Encryption method line
Content-length: [length]	CRLF	Data length line
. . . .		Arbitrary header line
CRLF		Header separator

Name	Description
Response line	<p>The protocol version, and a status code as well as string for the processing result for the request are set. Separate the elements with a blank space. For the protocol version, the protocol name and version number are separated by "/" and "ICP/2.0" are returned. The following status codes and strings are returned: 200 OK: Process successful. 500 NG: Process failed. An error occurred in the Authentication Module. When the process fails, see the information in the message body for the cause. (The response line shows only success or failure.)</p>

Name	Description
Encryption method line 10.0	<p>The communication message encryption library used to encrypt the body is set.</p> <p>Whether the communication message encryption library is for standard encryption or custom encryption determined as follows.</p> <ul style="list-style-type: none">•When libCryptoExit.sl for standard encryption is used to encrypt/decrypt the message body X-iw-encrypted header is set to "icewall." or The X-iw-encrypted header does not exist•When libExCryptoExit.sl for custom encryption is used to encrypt/decrypt the message body A value other than "icewall" is set in the X-iw-encrypted header to indicate custom encryption.
Data length line	The data length of the body is set.
Arbitrary header line 10.0	<p>An arbitrary header line sent when requesting and a header added by the Authentication Module (only when the UserExit routine is used) are set.</p> <p>The format is "header name: header value." (Example) Client-Id: 0001</p> <p>A line feed code is added at the end of the line to separate the header lines.</p> <p>The sent number of lines set is not defined (minimum zero). You cannot use the following header names because they are reserved words.</p> <div><ul style="list-style-type: none">•Content-length•Header beginning with "X-iw-"</div>
Header separator	A line feed code is set as the header separator.

3.2.3 Body

The body consists of the actual request and response data. This part is encrypted before transfer. The length of the encrypted data is the data length of the header.

Data 1	CRLF
Data 2	CRLF
...	
Data n	CRLF

In the body, necessary data lines are separated by a linefeed code.
A single line of data consists of the following elements.

Name	:	Blank space	Value
------	---	----------------	-------

- The order of data lines is not important.
- You can use any name for each data line, but you can use the following reserved words only for predefined purposes.

```
AGENT_ID
CERT_EXPR
CERT_NO
ERR_NO
IW_INFO
IW_PWD
IW_UID
MSG_ID
NEW_PWD
PWD_EXPR
REMOTE_ADDR
SOURCE_ADDR
SID_EXPR
ACC_URL
CERT_UID
METHOD
USER_AGENT
X-iw-encrypted
```

- The maximum number of elements is 512.
- The maximum size of each line (name and value) is 1024 bytes.
- A name and value are separated by a colon and blank space(s), and multiple blank spaces are treated as a single separator.
- When using a 2-byte character in data, convert it into hexadecimal as in "%xx."

Example) " 日 " → %93%FA

In the body section, you can set the following three types of information in addition to the elements necessary for each message:

- (1) Environmental information
- (2) Additional information
- (3) Supplemental information

(1) Environmental information

Information added to a request message for login is saved in the Authentication Module as the environmental information of the request source and can be used for configuring a user group. This information is returned to the request source in an access control response as a part of the user information. This information is valid until the user logs out.

(2) Additional information

Information added to a request message not related to login is treated as additional information for the request. This information is not saved in the Authentication Module and is deleted as soon as the response is returned.

(3) Supplemental information

Information added to a response message other than the necessary elements is treated as supplemental information. Supplemental information is not usually added to a response message but is returned to the request source when the UserExit routine for the Authentication Module adds it to the response message.

The following pages describe the format of each request and response message.

Login request using user ID and password

Overview	This login request is used when authenticating the user for the IceWall using the user ID and password only. Normal authentication uses this request.
Format	MSG_ID: ReqLoginUID _{CRLF} IW_UID: User ID _{CRLF} IW_PWD Password _{CRLF} REMOTE_ADDR: IP address _{CRLF} AGENT_ID: Request sender identification name _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following elements:</p> <p>MSG_ID Sets ReqLoginUID. This element is mandatory.</p> <p>IW_UID Sets the user ID of the user who logs in. The user ID length is 64 bytes or less. This element is mandatory.</p> <p>IW_PWD Sets the password of the user who logs in. The password length is 128 bytes or less. This element is mandatory.</p> <p>REMOTE_ADDR Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it is used in group control or request control for login. REMOTE_ADDR is stored as a part of the environmental information.</p> <p>AGENT_ID Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).</p> <p>USER_AGENT Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.</p> <p>Arbitrary element name You can send arbitrary information as the environmental information of the request sender. You can also use an arbitrary name. Information of this element is included in information inheritance</p>

notified to the Backend Web Server.

Sample usage

(1) When the user ID is "user01," password is "password," and the value for the environmental information "REQUEST_TIME" is set to "09:12:07" for login

MSG_ID: ReqLoginUID
IW_UID: user01
IW_PWD: password
REMOTE_ADDR: 192.168.0.1
AGENT_ID: IceWall SSO DFW
USER_AGENT: Mozilla/x.x
REQUEST_TIME: 09:12:07

(2) When logging in with the minimum required information

MSG_ID: ReqLoginUID
IW_UID: user01
IW_PWD: password

* The line feed code CRLF to separate data lines is omitted.

Remarks

- If you do not send REMOTE_ADDR with this request, the Authentication Module cannot perform grouping and request control using the IP address.
- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Forced login request using user ID and password

Overview	<p>When "duplicate login forbidden/exclusive login" is set in the Authentication Module, this request forces the users logging in with the same user ID to log out and then requires a single user to log in again. Usually, this request is used when "duplicate login error" is returned in the login response and forced login is required.</p>
Format	<p>MSG_ID: ReqForceLoginUID_{CRLF} IW_UID: User ID_{CRLF} IW_PWD Password_{CRLF} REMOTE_ADDR: IP address _{CRLF} AGENT_ID: Request sender identification name_{CRLF} USER_AGENT: User agent name_{CRLF} Arbitrary element name: Arbitrary data_{CRLF}</p>
Description	<p>This request consists of the following elements:</p> <p>MSG_ID Sets ReqForceLoginUID. This element is mandatory.</p> <p>IW_UID Sets the user ID of the user who logs in. The user ID length is 64 bytes or less. This element is mandatory.</p> <p>IW_PWD Sets the password of the user who logs in. The password length is 128 bytes or less. This element is mandatory.</p> <p>REMOTE_ADDR Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it is used in group control or request control for login. REMOTE_ADDR is stored as a part of the environmental information.</p> <p>AGENT_ID Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).</p> <p>USER_AGENT Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.</p> <p>Arbitrary element name You can send arbitrary information as the environmental information</p>

of the request sender . You can also use an arbitrary name.
Information of this element is included in information inheritance
notified to the Backend Web Server.

Sample usage (1) When the user ID is "user01," password is "password," and the value
for the environmental information "REQUEST_TIME" is set to
"09:12:07" for forced login
MSG_ID: ReqForceLoginUID
IW_UID: user01
IW_PWD: password
REMOTE_ADDR: 192.168.0.1
AGENT_ID: IceWall SSO DFW
USER_AGENT: Mozilla/x.x
REQUEST_TIME: 09:12:07

(2) When logging in with the minimum required information
MSG_ID: ReqForceLoginUID
IW_UID: user01
IW_PWD: password

* The line feed code CRLF to separate data lines is omitted.

Remarks • If you do not send REMOTE_ADDR with this request, the
Authentication Module cannot perform grouping and request control
using the IP address.

• If you do not send AGENT_ID with this request, the agent ID is not
output to the access log of the Authentication Module.

Login request using client certificate and password

Overview	This login request is used when authenticating the user for IceWall using the client certificate and password. This request is used when more secure authentication than the use of the user ID and password is required.
Format	MSG_ID: ReqLoginCERT _{CRLF} IW_UID: User ID _{CRLF} IW_PWD Password _{CRLF} CERT_NO: Serial number _{CRLF} CERT_EXPR: Expiration date _{CRLF} REMOTE_ADDR: IP address _{CRLF} AGENT_ID: Request sender identification name _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following elements:</p> <p>MSG_ID Sets ReqLoginCERT. This element is mandatory.</p> <p>IW_UID Sets the user ID contained in the client certificate of the user who logs in. The user ID length is 64 bytes or less. This element is mandatory.</p> <p>IW_PWD Sets the password of the user who logs in. The password length is 128 bytes or less. This element is mandatory.</p> <p>CERT_NO Sets the serial number of the client certificate. To improve the uniqueness of the certificate, you can specify "serial number:issuer." There is no limit on the data length. This element is mandatory.</p> <p>CERT_EXPR Sets the expiration date of the client certificate in the following format: YYYYMMDDhhmmss The length of the expiration date is 14 bytes or less. This element is mandatory.</p> <p>REMOTE_ADDR Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it is used in group control or request control for login. REMOTE_ADDR is stored as a part of the environmental</p>

information.

AGENT_ID

Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).

USER_AGENT

Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.

Arbitrary element name

You can send arbitrary information as the environmental information of the request sender. You can also use an arbitrary name. Information of this element is included in information inheritance notified to the Backend Web Server.

Sample usage

- (1) When the user ID is "user01," serial number is "01," and the client certificate with the expiration date of September 30 2009 is used for login
MSG_ID: ReqLoginCERT
IW_UID: user01
IW_PWD: password
CERT_NO: 01
CERT_EXPR: 20090930235959
REMOTE_ADDR: 192.168.0.1
AGENT_ID: IceWall SSO DFW
USER_AGENT: Mozilla/x.x
- (2) When the issuer of the client certificate is added to the serial number for login
MSG_ID: ReqLoginCERT
IW_UID: user01
IW_PWD: password
CERT_NO: 01:OU=Development/O=hp.com
CERT_EXPR: 20060930235959
REMOTE_ADDR: 192.168.0.1
AGENT_ID: IceWall SSO DFW
USER_AGENT: Mozilla/x.x
- (3) When logging in with the minimum required information
MSG_ID: ReqLoginCERT
IW_UID: user01
IW_PWD: password
CERT_NO: 01
CERT_EXPR: 20060930235959

* The line feed code CRLF to separate data lines is omitted.

Remarks

- If you do not send REMOTE_ADDR with this request, the Authentication Module cannot perform grouping and request control using the IP address.
- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Forced login request using client certificate and password

Overview	<p>When "duplicate login forbidden/exclusive login" is set in the Authentication Module, this request forces the users logging in with the same user ID to log out and then requires a single user to log in again. Usually, this request is used when "duplicate login error" is returned in the login response and forced login is required.</p>
Format	<p>MSG_ID: ReqForceLoginCERT_{CRLF} IW_UID: User ID_{CRLF} IW_PWD Password_{CRLF} CERT_NO: Serial number_{CRLF} CERT_EXPR: Expiration date_{CRLF} REMOTE_ADDR: IP address _{CRLF} AGENT_ID: Request sender identification name_{CRLF} USER_AGENT: User agent name_{CRLF} Arbitrary element name: Arbitrary data_{CRLF}</p>
Description	<p>This request consists of the following elements:</p> <p>MSG_ID Sets ReqForceLoginCERT. This element is mandatory.</p> <p>IW_UID Sets the user ID contained in the client certificate of the user who logs in. The user ID length is 64 bytes or less. This element is mandatory.</p> <p>IW_PWD Sets the password of the user who logs in. The password length is 128 bytes or less. This element is mandatory.</p> <p>CERT_NO Sets the serial number of the client certificate. To improve the uniqueness of the certificate, you can specify "serial number:issuer." There is no limit on the data length. This element is mandatory.</p> <p>CERT_EXPR Sets the expiration date of the client certificate in the following format: YYYYMMDDhhmmss The length of the expiration date is 14 bytes or less. This element is mandatory.</p> <p>REMOTE_ADDR Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it is used in group control or request control for</p>

login. REMOTE_ADDR is stored as a part of the environmental information.

AGENT_ID

Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).

USER_AGENT

Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.

Arbitrary element name

You can send arbitrary information as the environmental information of the request sender. You can also use an arbitrary name.

Information of this element is included in information inheritance notified to the Backend Web Server.

Sample usage

(1) When the user ID is "user01," serial number is "01," and the client certificate with the expiration date of September 30 2009 is used for forced login
MSG_ID: ReqForceLoginCERT
IW_UID: user01
IW_PWD: password
CERT_NO: 01
CERT_EXPR: 20090930235959
REMOTE_ADDR: 192.168.0.1
AGENT_ID: IceWall SSO DFW
USER_AGENT: Mozilla/x.x

* The line feed code CRLF to separate data lines is omitted.

Remarks

- If you do not send REMOTE_ADDR with this request, the Authentication Module cannot perform grouping and request control using the IP address.
- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Login response using user ID and password

Overview This response is returned from the Authentication Module for a login or forced login request using the user ID and password.

Format

When login is successful

MSG_ID: ResLoginUID_{CRLF}

ERR_NO: Login result_{CRLF}

IW_INFO: Session ID_{CRLF}

PWD_EXPR: Password expiration date_{CRLF}

SID_EXPR: Session ID expiration date _{CRLF}

Arbitrary element name: Arbitrary data_{CRLF}

When login fails

MSG_ID: ResLoginUID_{CRLF}

ERR_NO: Login result_{CRLF}

Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following elements:

MSG_ID
Sets ResLoginUID. This element is always set.

ERR_NO
Sets the login result. The following values are set. This element is always set.

Value	Description
0	Login successful
1	User ID Error
2	Maximum Login Count Error
3	Duplicate Login Error
4	Password Error
6	Login Stop Error
7	No Group Error
8	Concurrent Usage Error
9	Account Lock Error
15	Login successful and password change request due to expired password
18	Authentication DB Down Error
19	Request Execution Permission Error
20	Login successful within password expiration warning period
21	Session ID Generation Error
91	User-Defined Error 1
92	User-Defined Error 2

Value	Description
93	User-Defined Error 3
99	System Error

IW_INFO

Sets the session ID. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

When the cookie expiration date addition flag is enabled (COOKIEEXP=1), the expiration date is appended to the session ID, separated with a semicolon, in the following format:

xxx - xxx; Expires=Wdy, DD-Mon-YYYY HH:MM:SS GMT;

PWD_EXPR

Sets the password expiration date of the login user. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

However, it is not sent unless the password expiration date information is obtained by the Authentication Module.

SID_EXPR

Sets the timeout time of the session ID of the login user. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

The expiration time set here is the time calculated upon login. The session may not time out at this time because the expiration time stored in the Authentication Module may change due to the module setting.

This data is sent regardless of the setting of the Authentication Module.

Arbitrary element name

The supplemental information added by the UserExit routine for the Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent regardless of whether or not the login is successful.

Sample usage

- (1) When a login using the user ID and password is successful and the cookie expiration date addition flag is enabled
- MSG_ID: ResLoginUID
ERR_NO: 0
IW_INFO: xxx-xxx; Expires=Sun, 14-Dec-2008 23:59:59 GMT;
PWD_EXPR: 20080930172959
SID_EXPR: 20081214235959

- (2) When a login using the user ID and password is successful but the password is within the expiration warning period

MSG_ID: ResLoginUID

ERR_NO: 20

IW_INFO: xxx-xxx

PWD_EXPR: 20080930172959

SID_EXPR: 20081214235959

- (3) When the login fails due to a password error (with supplemental information)

MSG_ID: ResLoginUID

ERR_NO: 4

REQ_NO: 1073

* The line feed code CRLF to separate data lines is omitted.

Remarks

- For access control (using the user ID and password), password change, and logout requests, use the IW_INFO data value sent by this request as the session ID.
- If the expiration date is added to the IW_INFO value, use the string preceding the semicolon as the session ID.

Response for login request using client certificate and password

Overview This response is returned from the Authentication Module for a login or forced login request using the client certificate and password.

Format When login is successful
MSG_ID: ResLoginCERT_{CRLF}
ERR_NO: Login result_{CRLF}
IW_INFO: Session ID_{CRLF}
PWD_EXPR: Password expiration date_{CRLF}
SID_EXPR: Session ID expiration date _{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

When login fails
MSG_ID: ResLoginCERT_{CRLF}
ERR_NO: Login result_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following elements:
MSG_ID
Sets ResLoginCERT. This element is always set.
ERR_NO
Sets the login result. The following values are set. This element is always set.

Value	Description
0	Login successful
1	User ID Error
2	Maximum Login Count Error
3	Duplicate Login Error
4	Password Error
5	Certificate Serial Number Error
6	Login Stop Error
7	No Group Error
9	Account Lock Error
15	Login successful and password change request due to expired password
18	Authentication DB Down Error
19	Request Execution Permission Error
20	Login successful within password expiration warning period
21	Session ID Generation Error
91	User-Defined Error 1

Value	Description
92	User-Defined Error 2
93	User-Defined Error 3
99	System Error

IW_INFO

Sets the session ID. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

When the cookie expiration date addition flag is enabled (COOKIEEXP=1), the expiration date is appended to the session ID, separated with a semicolon, in the following format:

xxx - xxx; Expires=Wdy, DD-Mon-YYYY HH:MM:SS GMT;

PWD_EXPR

Sets the password expiration date of the login user. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

However, it is not sent unless the password expiration date information is obtained by the Authentication Module.

SID_EXPR

Sets the timeout time of the session ID of the login user. This element is set only when the login result is "Login successful," "Login successful and password change request due to expired password," or "Login successful within password expiration warning period."

The expiration time set here is the time calculated upon login. The session may not time out at this time because the expiration time stored in the Authentication Module may change due to the module setting.

This data is sent regardless of the setting of the Authentication Module.

Arbitrary element name

The supplemental information added by the UserExit routine for the Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent regardless of whether or not login is successful.

Sample usage

- (1) When a login using the client certificate and password is successful and the cookie expiration date addition flag is enabled
- MSG_ID: ResLoginCERT
ERR_NO: 0
IW_INFO: xxx-xxx; Expires=Sun, 14-Dec-2008 23:59:59 GMT;
PWD_EXPR: 20090930172959
SID_EXPR: 20081214235959

- (2) When a login using the client certificate and password is successful but the password has expired

MSG_ID: ResLoginCERT

ERR_NO: 15

IW_INFO: xxx-xxx

PWD_EXPR: 20081220172959

SID_EXPR: 20081221235959

- (3) When the login fails because the maximum login count is exceeded

MSG_ID: ResLoginCERT

ERR_NO: 2

* The line feed code CRLF to separate data lines is omitted.

Remarks

- For access control (using the client certificate and password), password change, and logout requests, use the IW_INFO data value sent by this request as the session ID.
- If the expiration date is added to the IW_INFO value, use the string preceding the semicolon as the session ID.

Access control request (user ID and password)

Overview	This request is used when requesting the Authentication Module to check for the access permission for a URL. It is also used when a login using the user ID and password is performed.
Format	MSG_ID: ReqAccessUID _{CRLF} IW_INFO: Session ID _{CRLF} ACC_URL: Access permission check URL _{CRLF} AGENT_ID: Request sender identification name _{CRLF} REMOTE_ADDR: Client IP address _{CRLF} METHOD: Method name _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following data:</p> <p>MSG_ID Sets ReqAccessUID. This element is mandatory.</p> <p>IW_INFO Sets the value of the IW_INFO element sent in "Response for login request using user ID and password." This element is mandatory. The session ID length is 32 or 64 bytes depending on the setting of the SESSIONIDLEN parameter.</p> <p>ACC_URL Sets the URL to check for the access permission. Specify the URL as an absolute path. The URL length is 1024 bytes or less. This element is mandatory. The URL can be URL-encoded or not.</p> <p>AGENT_ID Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).</p> <p>REMOTE_ADDR Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it may be used in the future extension.</p> <p>METHOD Sets the method requested by the client. It is recommended to set this element because it may be used in the future extension.</p> <p>USER_AGENT Sets the user agent of the browser used by the client. It is</p>

recommended to set this element because it may be used in the future extension.

Arbitrary element name

The request sender can send arbitrary information as the additional information that is only valid for the request. You can also use an arbitrary name. The information sent here is sent to the Authentication Module and then deleted when the response message is received.

Sample usage

(1) For access control request for the URL "http://www.hp.com/"

```
MSG_ID: ReqAccessUID
IW_INFO: xxx-xxx
ACC_URL: http://www.hp.com/
AGENT_ID: IceWall SSO DFW
REMOTE_ADDR: 192.168.0.1
METHOD: GET
USER_AGENT: Mozilla/x.x
```

* The line feed code CRLF to separate data lines is omitted.

Remarks

- The URL specified in the ACC_URL element is used for access permission check without modification. Note that if the case used in the URL is not correctly specified in the access control file for the Authentication Module, the result returned may be different from expected.
- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Access control request (client certificate and password)

Overview	This request is used when requesting the Authentication Module to check for the access permission for a URL. It is also used when a login using the client certificate and password is performed.
Format	MSG_ID: ReqAccessCERT _{CRLF} IW_INFO: Session ID _{CRLF} ACC_URL: Access permission check URL _{CRLF} CERT_UID: User ID _{CRLF} CERT_NO: Serial number _{CRLF} CERT_EXPR: Expiration date _{CRLF} AGENT_ID: Request sender identification name _{CRLF} REMOTE_ADDR: Client IP address _{CRLF} METHOD: Method name _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following data:</p> <p>MSG_ID Sets ReqAccessCERT. This element is mandatory.</p> <p>IW_INFO Sets the value of the IW_INFO element sent in "Response for login request using client certificate and password." This element is mandatory. The session ID length is 32 or 64 bytes depending on the setting of the SESSIONIDLEN parameter.</p> <p>ACC_URL Sets the URL to check for the access permission. Specify the URL as an absolute path. There is no limit on the URL length. This element is mandatory. The URL can be URL-encoded or not.</p> <p>CERT_UID Sets the user ID contained in the client certificate of the user who is controlled for access. Specify the same user ID as specified for login. This element is mandatory. However, if the certificate check flag for access control (ACCCTRLFLG) is set to 1 or a higher number, access control is correctly performed without this setting.</p> <p>CERT_NO Sets the serial number contained in the client certificate of the user for whom access control is to be performed. Specify the same serial number as specified for login. It is recommended to set this element</p>

because it may be used in the future extension.

CERT_EXPR

Sets the expiration date contained in the client certificate of the user who is controlled for access in the following format:

YYYYMMDDhhmmss

Specify the same expiration date as specified for login. The length of expiration date is 14 bytes or less. It is recommended to set this element because it may be used in the future extension.

AGENT_ID

Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).

REMOTE_ADDR

Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it may be used in the future extension.

METHOD

Sets the method requested by the client. It is recommended to set this element because it may be used in the future extension.

USER_AGENT

Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.

Arbitrary element name

The request sender can send arbitrary information as the additional information that is only valid for the request. You can also use an arbitrary name. The information sent here is sent to the Authentication Module and then deleted when the response message is received.

Sample usage

(1) For access control request for the URL "http://www.hp.com/"
MSG_ID: ReqAccessCERT
IW_INFO: xxx-xxx
ACC_URL: http://www.hp.com/
CERT_UID: user01
CERT_NO: 01
CERT_EXPR: 20090930235959
AGENT_ID: IceWall SSO DFW
REMOTE_ADDR: 192.168.0.1
METHOD: GET
USER_AGENT: Mozilla/x.x

* The line feed code CRLF to separate data lines is omitted.

Remarks

- The URL specified in the ACC_URL element is used for access permission check without modification. Note that if the case used in the URL is not correctly specified in the access control file for the Authentication Module, the result returned may be different from expected.
- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Access control response (user ID and password)

Overview This response is returned for an access control request using the user ID and password.

Format

Access allowed
MSG_ID: ResAccessUID_{CRLF}
ERR_NO: Access control result_{CRLF}
IW_UID: User ID_{CRLF}
IW_PWD Password_{CRLF}
Authentication DB column: User information_{CRLF}
Environment information: Environment information_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Access control error
MSG_ID: ResAccessUID_{CRLF}
ERR_NO: Access control result_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following data:

MSG_ID
Sets ResAccessUID. This element is always set.

ERR_NO
The access control result is set. The following values are set. This element is always set.

Value	Description
0	Access allowed
10	Access Permission Error
11	Relogin request due to login timeout
15	Password change request due to expired password
19	No Request Execution Permission Error
91	User-Defined Error 1
92	User-Defined Error 2
93	User-Defined Error 3
99	System Error

IW_UID

The user ID specified for login is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

IW_PWD

The password specified for login is set. This element is set only when

the access control result is "Access allowed" and the Authentication Module is set to send this element.

Authentication DB column

The column of the user information registered in the Authentication DB and specified to read by the Authentication Module is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

Environment information

The entire information specified as the environmental information when the user logs in is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

Arbitrary element name

The supplemental information added by the UserExit routine for the Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent regardless of whether or not access is successful.

Sample usage

(1) When the access control result is "Access allowed"

MSG_ID: ResAccessUID
ERR_NO: 0
IW_UID: user01
IW_PWD: password
USERID: user01
PASSWD: {MD5}xxx
LOGINDATE: 20081214091708
REMOTE_ADDR: x.x.x.x

(2) When the access control result is other than "Access allowed"

MSG_ID: ResAccessUID
ERR_NO: 10

* The line feed code CRLF to separate data lines is omitted.

Remarks

None

Access control response (client certificate and password)

Overview This response is returned for an access control request using the client certificate and password.

Format

Access allowed

MSG_ID: ResAccessCERT_{CRLF}

ERR_NO: Access control result_{CRLF}

IW_UID: User ID_{CRLF}

IW_PWD: Password_{CRLF}

Authentication DB column: User information_{CRLF}

Environment information name: Environment information_{CRLF}

Arbitrary element name: Arbitrary data_{CRLF}

Access control error

MSG_ID: ResAccessCERT_{CRLF}

ERR_NO: Access control result_{CRLF}

Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following data:

MSG_ID
Sets ResAccessCERT. This element is always set.

ERR_NO
The access control result is set. The following values are set. This element is always set.

Value	Description
0	Access allowed
10	Access Permission Error
11	Relogin request due to login timeout
15	Password change request due to expired password
19	No Request Execution Permission Error
91	User-Defined Error 1
92	User-Defined Error 2
93	User-Defined Error 3
99	System Error

IW_UID
The user ID specified for login or stored in the client certificate is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

IW_PWD
The password specified for login is set. This element is set only when

the access control result is "Access allowed" and the Authentication Module is set to send this element.

Authentication DB column

The column of the user information registered in the Authentication DB and specified to read by the Authentication Module is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

Environment information

The entire information specified as the environmental information when the user logs in is set. This element is set only when the access control result is "Access allowed" and the Authentication Module is set to send this element.

Arbitrary element name

The supplemental information added by the UserExit routine for the Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent whether access succeeds or fails.

Sample usage	(1) When the access control result is "Access allowed"
	MSG_ID: ResAccessCERT ERR_NO: 0 IW_UID: user01 IW_PWD: password USERID: user01 PASSWD: {MD5}xxx RASERIAL: 123 IWSERIAL: 123 REMOTE_ADDR: x.x.x.x
	(2) When the access control result is other than "Access allowed"
	MSG_ID: ResAccessCERT ERR_NO: 10

* The line feed code CRLF to separate data lines is omitted.

Remarks	None
---------	------

Password change request

Overview	This request is used to change the password.
Format	MSG_ID: ReqPasswdChg _{CRLF} IW_INFO: Session ID _{CRLF} NEW_PWD: Password _{CRLF} PWD_EXPR: Password expiration date _{CRLF} CERT_UID: User ID _{CRLF} CERT_NO: Serial number _{CRLF} CERT_EXPR: Expiration date _{CRLF} AGENT_ID: Request sender identification name _{CRLF} REMOTE_ADDR: Client IP address _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following data:</p> <p>MSG_ID Sets ReqPasswdChg. This element is mandatory.</p> <p>IW_INFO Sets the value of the IW_INFO element sent in "Response for login request using user ID and password" or "Response for login request using client certificate and password." This element is mandatory. The session ID length is 32 or 64 bytes, depending on the setting of the SESSIONIDLEN parameter.</p> <p>NEW_PWD Sets the new password. The password length is 128 bytes or less. This element is mandatory.</p> <p>PWD_EXPR Sets the expiration period of the new password. Specify the expiration period as the number of days. This element is mandatory.</p> <p>CERT_UID Sets the user ID contained in the client certificate of the user whose password is to be changed, only when the client certificate is used. Specify the same user ID as specified for login. The user ID length is 64 bytes or less. It is recommended to set this element because it may be used in the future extension.</p> <p>CERT_NO Sets the serial number contained in the client certificate of the user whose password is to be changed only when the client certificate is used. Specify the same serial number as specified for login. It is recommended to set this element because it may be used in the future</p>

extension.

CERT_EXPR

Sets the expiration date contained in the client certificate of the user whose password is to be changed in the following format only when the client certificate is used:

YYYYMMDDhhmmss

Specify the same expiration date as specified for login. The length of expiration date is 14 bytes or less. It is recommended to set this element because it may be used in the future extension.

AGENT_ID

Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).

REMOTE_ADDR

Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it may be used in the future extension.

USER_AGENT

Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.

Arbitrary element name

The request sender can send arbitrary information as the additional information that is valid only during the request. You can also use an arbitrary name. The information sent here is notified to the Authentication Module and then deleted when the response message is received.

Sample usage	(1) When the new password is "passwd" and the expiration period is 30 days MSG_ID: ReqPasswdChg IW_INFO: xxx-xxx NEW_PWD: passwd PWD_EXPR: 30 AGENT_ID: IceWall SSO DFW REMOTE_ADDR: 192.168.0.1 USER_AGENT: Mozilla/x.x
--------------	---

* The line feed code CRLF to separate data lines is omitted.

Remarks	<ul style="list-style-type: none">• If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.
---------	---

Password change response

Overview Response for a password change request.

Format

Password change successful
MSG_ID: ResPasswdChg_{CRLF}
ERR_NO: Password change result_{CRLF}
PWD_EXPR: Password expiration date_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Password change failed
MSG_ID: ResPasswdChg_{CRLF}
ERR_NO: Password change result_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following data:

MSG_ID
Sets. ResPasswdChg. This element is always set.

ERR_NO
Sets the password change result. The following values are set. This element is always set.

Value	Description
0	Password change successful
13	Password Change Error
14	Password Change Permission Error
16	Password Policy Error
18	Authentication DB Down Error
19	No Request Execution Permission Error
22	Minimum Password Length Error
23	Maximum Password Length Error
24	Password Character Type Error
25	Same Password as User ID Error
91	User-Defined Error 1
92	User-Defined Error 2
93	User-Defined Error 3
99	System Error

PWD_EXPR
The expiration period of the new password is set. This element is set only when the password change is successful.

Arbitrary element name
The supplemental information added by the UserExit routine for the

Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent regardless of whether or not password change is successful.

Sample usage (1) When password change is successful

MSG_ID: ResPasswdChg

ERR_NO: 0

PWD_EXPR: 20090101000000

(2) When password change fails

MSG_ID: ResPasswdChg

ERR_NO: 16

* The line feed code CRLF to separate data lines is omitted.

Remarks None

Logout request

Overview	This request is used to log out the user.
Format	MSG_ID: ReqLogout _{CRLF} IW_INFO: Session ID _{CRLF} CERT_UID: User ID _{CRLF} CERT_NO: Serial number _{CRLF} CERT_EXPR: Expiration date _{CRLF} AGENT_ID: Request sender identification name _{CRLF} REMOTE_ADDR: Client IP address _{CRLF} USER_AGENT: User agent name _{CRLF} Arbitrary element name: Arbitrary data _{CRLF}
Description	<p>This request consists of the following data:</p> <p>MSG_ID Sets ReqLogout. This element is mandatory.</p> <p>IW_INFO Sets the value of the IW_INFO element sent in "Response for login request using user ID and password" or "Response for login request using client certificate and password." This element is mandatory. The session ID length is 32 or 64 bytes, depending on the setting of the SESSIONIDLEN parameter.</p> <p>CERT_UID Sets the user ID contained in the client certificate of the user who is to be logged out, only when the client certificate is used. Specify the same user ID as specified for login. The user ID length is 64 bytes or less. It is recommended to set this element because it may be used in the future extension.</p> <p>CERT_NO Sets the serial number contained in the client certificate of the user who is to be logged out, only when the client certificate is used. Specify the same serial number as specified for login. It is recommended to set this element because it may be used in the future extension.</p> <p>CERT_EXPR Sets the expiration date contained in the client certificate of the user who is to be logged out in the following format, only when the client certificate is used: YYYYMMDDhhmmss Specify the same expiration date as specified for login. The length of the expiration date is 14 bytes or less. It is recommended to set this element because it may be used in the future extension.</p>

AGENT_ID

Sets the name to identify the requester. There is no limit on the length of the identification name. It is recommended to set this element because it is output to the access log of the Authentication Module (depending on the setting of the module).

REMOTE_ADDR

Sets the client IP address. Use the dot-separated format for the IPv4 address, and specify a 128-bit address consisting of eight 16-bit fields of hexadecimal values for the IPv6 address. It is recommended to set this element because it may be used in the future extension.

USER_AGENT

Sets the user agent of the browser used by the client. It is recommended to set this element because it may be used in the future extension.

Arbitrary element name

The request sender can send arbitrary information as the additional information that is valid only during the request. You can also use an arbitrary name. The information sent here is notified to the Authentication Module and then deleted when the response message is received.

Sample usage

(1) When logging out
MSG_ID: ReqLogout
IW_INFO: xxx-xxx
AGENT_ID: IceWall SSO DFW
REMOTE_ADDR: 192.168.50.1
USER_AGENT: Mozilla/x.x

* The line feed code CRLF to separate data lines is omitted.

Remarks

- If you do not send AGENT_ID with this request, the agent ID is not output to the access log of the Authentication Module.

Logout response

Overview Response for a logout request.

Format

Logout successful
MSG_ID: ResLogout_{CRLF}
ERR_NO: Logout result_{CRLF}
IW_INFO: Session ID_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Logout failed
MSG_ID: ResLogout_{CRLF}
ERR_NO: Logout result_{CRLF}
Arbitrary element name: Arbitrary data_{CRLF}

Description This request consists of the following data:

MSG_ID
A response is set. It is fixed to "ResLogout." This element is always set.

ERR_NO
The logout result is set. The following values are set. This element is always set.

Value	Description
0	Logout successful
17	Logout Error
19	No Request Execution Permission Error
99	System Error

IW_INFO
The session ID and past date for logout are set in the following format. This element is set only when the logout result is "Logout successful."
xxx-xxx; Expires=Thu, 01-Jan-1970 09:00:01 GMT;

Arbitrary element name
The supplemental information added by the UserExit routine for the Authentication Module is set. There is no limit on the number of elements. Note that if supplemental information exists, it is sent regardless of whether logout is successful or failed.

Sample usage (1) When logout is successful
MSG_ID: ResLogout
ERR_NO: 0
IW_INFO: xxx-xxx; Expires=Thu, 01-Jan-1970 09:00:01 GMT;

(2) When logout fails

MSG_ID: ResPasswdChg

ERR_NO: 17

* The line feed code CRLF to separate data lines is omitted.

Remarks	None
---------	------

4 Communication Message Encryption Library 10.0

This chapter describes the specifications of the communication message encryption library.

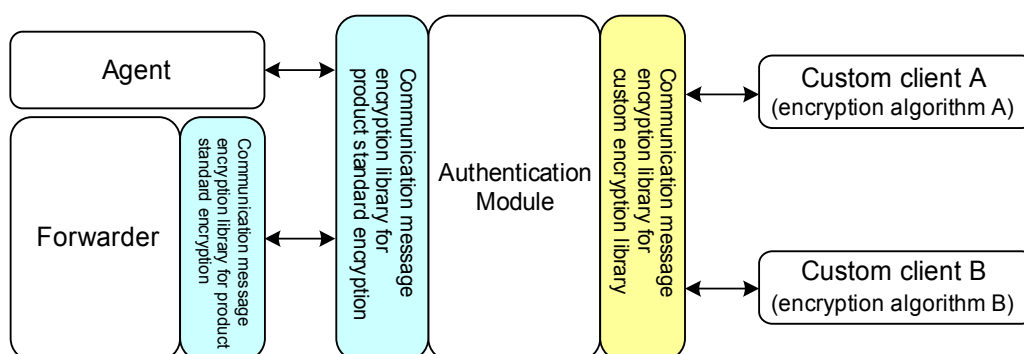
4.1 What is communication message encryption library?

The communication message encryption library encrypts and decrypts communication messages using ICP 2.0 transferred between the modules of the IceWall SSO.

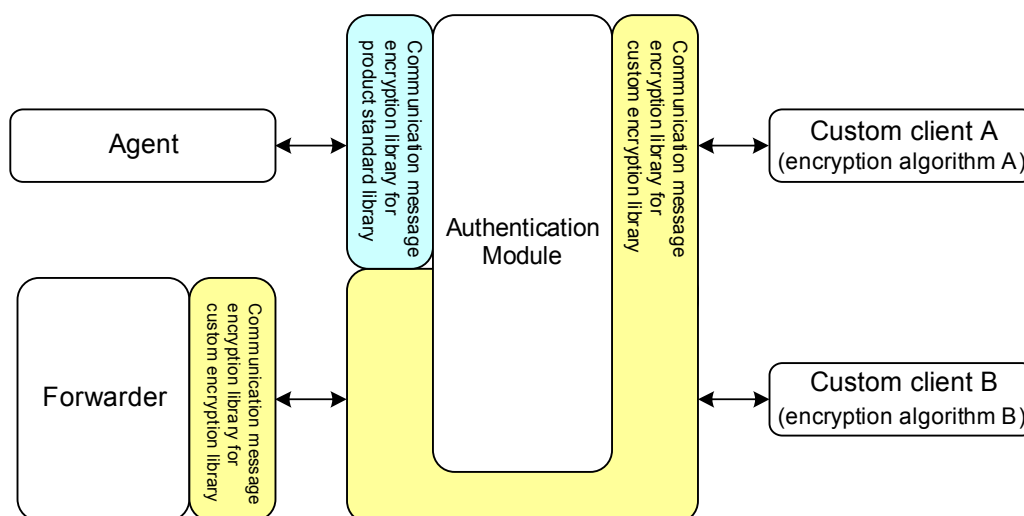
4.2 Extending the communication message encryption library

The communication message encryption library for ICP 2.0 up to 8.0 R3 had to be a single encryption algorithm for all modules and custom clients using ICP 2.0. From this version, you can use a communication message encryption library for custom encryption in addition to the encryption library for product standard encryption, allowing you to use the product standard and custom encryption algorithms concurrently.

- (1) When communicating between a custom client and Authentication Module using the customized encryption algorithm



- (2) When communicating between the Forwarder, custom client, and Authentication Module using a customized encryption algorithm



4.2.1 Communication message encryption library for product standard encryption

ICP 2.0 communication message encryption library used by each IceWall SSO product to communicate with the Authentication Module.

As with Version 8.0 R3 or earlier, you can also develop a communication message encryption library for product standard encryption.

4.2.2 Communication message encryption library for custom encryption

The ICP 2.0 communication message encryption library for custom encryption. You can use a custom encryption algorithm for communication between the Authentication Module and ICP 2.0, without affecting ICP 2.0 communication using the communication message encryption library for the product standard encryption. When the value of the X-iw-encrypted field in the ICP 2.0 request header is other than "icewall," the system judges that the request uses the custom encryption algorithm and uses the communication message encryption library for the custom encryption to encrypt/decrypt the request.

To encrypt/decrypt a request using the communication message encryption library for custom encryption, you need to add information indicating that the request was encrypted using the library to the ICP 2.0 request header.

- (1) For a login request using ICP 2.0 user ID and password encrypted by custom algorithm from custom client A

REQ / ICP/2.0
X-iw-encrypted: encryptA
Content-length: 55

MSG_ID: ReqLoginUID
IW_UID: UserID
IW_PWD: Password

* In fact, the body is encrypted using the communication message encryption library for custom encryption.

- (2) For a login request using ICP 2.0 user ID and password encrypted by custom algorithm from custom client B

REQ / ICP/2.0
X-iw-encrypted: encryptB
Content-length: 55

MSG_ID: ReqLoginUID
IW_UID: UserID
IW_PWD: Password

* In fact, the body is encrypted using the communication message encryption library for custom encryption.

4.3 Development of the communication message encryption library

For the product standard communication message encryption library, you can change the encryption and decryption algorithm to an arbitrary one.

For a custom communication message encryption library, apart from the communication message encryption library for product standard encryption, you can use a custom encryption and decryption algorithm as the communication message encryption library.

From this version, when you develop a communication message encryption library using ICP 2.0 with a custom encryption algorithm, it is recommended to develop a communication message encryption library for custom encryption to avoid the custom algorithm from affecting the communication message encryption library for the product standard encryption.

4.3.1 Development procedure of the communication message encryption library for product standard encryption

The following describes the general development procedure for changing the communication message encryption library for product standard encryption to an arbitrary encryption/decryption algorithm.

- (1) Determining the encryption/decryption specifications
Determine the encryption and decryption algorithm.

- (2) Creating a library
Program the source code according to the determined algorithm. Create the communication message encryption library as a shared library. Be sure to link the archive file containing the API functions (libctp.a) when developing the library.

Use the skeleton source included in the development kit for coding.
Build the code using the makefile included in the development kit for the Forwarder or Authentication Module.

Forwarder

```
$ cd /opt/icewall-ssso/developkit/dfw/DfwCryptoExit  
$ make -f Makefile
```

Authentication Module

```
$ cd /opt/icewall-ssso/developkit/certd/CryptoExit  
$ make -f Makefile
```

- (3) Testing and debugging
Test and debug the created shared library for correct operation using a test program. After the shared library has passed the test, integrate it with the Forwarder or the Authentication Module and then perform an operation test.

Install the communication message encryption library for the Forwarder in the following procedure.

- (1) Back up the communication message encryption library for product standard encryption installed by default.

```
$ cd /opt/icewall-ssso/lib  
$ cp -p libDfwCryptoExit.sl libDfwCryptoExit.sl.bak
```

- (2) Copy the developed communication message encryption library for product standard encryption to the target location.

```
$ cp -p /opt/icewall-ssso/developkit/dfw/DfwCryptoExit/libDfwCryptoExit.sl /opt/icewall-ssso/lib
```

Install the communication message encryption library for the Authentication Module in the following procedure.

- (1) Stop the Authentication Module.

```
$ /opt/icewall-ssso/certd/bin/end-cert
```

- (2) Back up the communication message encryption library for product standard encryption installed by default.

```
$ cd /opt/icewall-ssso/lib
$ cp -p libCryptoExit.sl libCryptoExit.sl.bak
```

- (3) Copy the developed communication message encryption library for product standard encryption to the target location.

```
$ cp -p /opt/icewall-ssso/developkit/certd/CryptoExit/libCryptoExit.sl /opt/icewall-ssso/lib
```

- (4) Start the Authentication Module.

```
$ /opt/icewall-ssso/certd/bin/start-cert
```

4.3.2 Development procedure of the communication message encryption library for custom encryption

The following describes the general development procedure for using the communication message encryption library for custom encryption.

- (1) Determining the encryption/decryption specifications
Determine the encryption and decryption algorithm.

- (2) Creating a library
Program the source code according to the determined algorithm. Create the communication message encryption library as a shared library. Be sure to link the archive file containing the API functions (libctp.a) when developing the library.

Use the skeleton source included in the development kit for coding.
Build the code using the makefile included in the development kit.

```
$ cd /opt/icewall-ssso/developkit/certd/ExCryptoExit
$ make -f Makefile
```

- (3) Testing and debugging

Test and debug the created shared library for correct operation using a test program. After the shared library has passed the test, integrate it with the Forwarder or the Authentication Module and then perform an operation test.

Install the communication message encryption library for the Authentication Module with the following procedure.

- (1) Stop the Authentication Module.

```
$ /opt/icewall-ssso/certd/bin/end-cert
```

- (2) Back up the communication message encryption library for custom encryption installed by default.

```
$ cd /opt/icewall-ssso/lib
$ cp -p libExCryptoExit.sl libExCryptoExit.sl.bak
```

- (3) Copy the developed communication message encryption library for custom encryption to the target location.

```
$ cp -p /opt/icewall-ssso/developkit/certd/ExCryptoExit/libExCryptoExit.sl /opt/icewall-ssso/lib
```

- (4) Start the Authentication Module.

```
$ /opt/icewall-ssso/certd/bin/start-cert
```

4.4 Notes on using the communication message encryption library for custom encryption

When the Forwarder and Authentication Module communicate with each other using a custom encryption method via ICP 2.0, follow the instructions below.-

- Create and install the communication message encryption library with a custom encryption algorithm using the development kit for the communication message encryption library for standard encryption for the Forwarder.
- Set a value other than "icewall" to the ICP_ENCSTR parameter.

Example) When notifying from the Forwarder to the Authentication Module that the custom encryption method "encrypt01" is used via ICP 2.0

```
ICP_ENCSTR=encrypt01
```

5 Specifications of the Communication Message Encryption Library for Product Standard Encryption

This chapter describes the specifications of the communication message encryption library for product standard encryption.

Callback functions that encrypt and decrypt a message and then return data to the caller, as well as API functions available within the callback functions, are provided.

5.1 Callback functions

The following two callback functions are available.

Function name	Process
IW_ExEncrypto()	Encrypts communication messages with product standard encryption.
IW_ExDecrypto()	Decrypts communication messages with product standard encryption.

Each callback function returns the address of the encryption structure. The following gives the specifications of the structure.

Structure name: IW_EXCRYPTO

Variable name	Type	Description
buff	char*	Stores the address of the area where the encrypted or decrypted communication message data is stored.
buflen	int	Stores the length of encrypted or decrypted communication message data.
result	int	Stores the encryption or decryption result of the API.
errmsg	char*	If an error occurs in encryption or decryption, stores an error message that notifies the error to the caller.

The following pages describe the specifications of each callback function.

IW_ExEncrypto

The following gives the specifications of the callback function that encrypts communication messages.

Format	IW_EXCRYPTO *IW_ExEncrypto(char* buff, int buflen)
Argument	buff :Address of the area where the message to be encrypted is stored. buflen :Message data length.
Return value	The function returns one of the following values. Other than NULL: Encryption successful or failed (with error message) NULL: Encryption failed (without error message)
Description	<p>Perform the following in this callback function.</p> <ol style="list-style-type: none">(1) Message encryption(2) Storage of encrypted message to the encryption structure <p>The operation of the Authentication Module changes as follows depending on the value set to each member of the encryption structure indicated by the return value.</p> <ul style="list-style-type: none">• buff \neq NULL and buflen \geq 1 and result = 0 Encryption successful. errmsg is ignored.• Other than successful (buff = NULL or buflen < 1 or result \neq 0) The Authentication Module judges that encryption has failed and outputs the result and errmsg values as messages to the error log.
Restrictions	<p>For the return value of this callback function, do not set a value other than the address of the encryption structure obtained by IW_ExCPTCreateCrypto() mentioned below or NULL.</p> <p>The argument of this callback function is for reference only. Do not change the value.</p>

IW_ExDecrypto

The following gives the specifications of the callback function that decrypts communication messages.

Format	IW_EXCRYPTO *IW_ExDecrypto(char* buff, int buflen)
Argument	buff :Address of the area where the message to decrypt is stored. buflen :Message data length.
Return value	The function returns one of the following values. Other than NULL: Decryption successful or failed (with error message) NULL: Decryption failed (without error message)
Description	<p>Perform the following in this callback function.</p> <ol style="list-style-type: none">(1) Message decryption(2) Storage of decrypted message to the encryption structure <p>The operation of the Authentication Module changes as follows depending on the value set to each member of the encryption structure indicated by the return value.</p> <ul style="list-style-type: none">• buff \neq NULL and buflen \geq 1 and result = 0 Decryption successful. errmsg is ignored.• Other than successful (buff = NULL or buflen < 1 or result \neq 0) The Authentication Module judges that decryption has failed and outputs the result and errmsg values as messages to the error log.
Restrictions	<p>For the return value of this callback function, do not set a value other than the address of the encryption structure obtained by IW_ExCPTCreateCrypto() mentioned below or NULL.</p> <p>The argument of this callback function is for reference only. Do not change the value.</p>

5.2 API function

The following API functions can be called within the callback functions.

API name	Process
IW_ExCPTCreateCrypto()	Creates the encryption structure.
IW_ExCPTReleaseCrypto()	Releases the encryption structure.

The following pages describe the specifications of each API function.

IW_ExcPTCreateCrypto

The following gives the specifications of the API function for creating the encryption structure to be referenced in the return value of the callback function.

Format **IW_EXCRYPTO *IW_ExcPTCreateCrypto(char* buff, int len, int result, char* errmsg)**

Argument **buff** :Specify the address of the area where the message to be encrypted (or decrypted) that is returned from the callback function is stored.

len :Specify the data length of the message.

result :Specify 0 to terminate the callback function as successful. Specify an error code to terminate it as failed. You can specify an arbitrary value other than "0" for the error code.

errmsg :Specify the address of the area where the error message returned when the callback function is terminated as failed is stored.

Return value The function returns one of the following values.
Other than NULL: Address of the encryption structure
NULL: Value could not be obtained due to an internal error

Restrictions If the len argument value is less than the quantity of data stored in the buffer specified by the buff argument, data up to the length specified by len is stored.

If the len argument value is negative, no data is stored even if data exists in the buffer specified by the buff argument.

Even when the buff and len argument values are NULL and negative, respectively, if the result and/or errmsg arguments are specified, they are stored.

If the errmsg argument value exceeds 200 characters, the error message may be truncated to 200 characters when the Authentication Module outputs it to the error log.

When you do not use the area obtained by this API as the return value of the callback function, call IW_ExcPTReleaseCrypto() to destroy the area.

IW_ExCPTReleaseCrypto

The following gives the specifications of the API function for releasing the encryption structure created by IW_ExCPTCreateCrypto().

Format	int IW_ExCPTReleaseCrypto (IW_EXCRYPTO* excrypto)
Argument	excrypto : Address of the encryption structure.
Return value	The function returns one of the following values. 0: Release successful Other than 0: Release failed
Restrictions	For the argument of this API, do not specify a value other than the address of the encryption structure obtained by IW_ExCPTCreateCrypto().

6 Library Specifications of the Communication Message Encryption Library for Custom Encryption **10.0**

This chapter describes the specifications of the communication message encryption library for custom encryption.

Callback functions that encrypt and decrypt a message and then return data to the caller, as well as API functions available within the callback functions, are provided.

6.1 Callback functions

The following two callback functions are available.

Function name	Process
CS_ExEncrypto()	Encrypts communication messages with custom encryption.
CS_ExDecrypto()	Decrypts communication messages with custom encryption.

Each callback function returns the address of the encryption structure. The following gives the specifications of the structure.

Structure name: IW_EXCRYPTO

Variable name	Type	Description
req	CS_REQ_DATA*	Stores the address of the area where the communication message data from the request information is stored.
buff	char*	Stores the address of the area where the encrypted or decrypted communication message data is stored.
buflen	int	Stores the length of encrypted or decrypted communication message data.

The following pages describe the specifications of each callback function.

CS_ExEncrypto

The following gives the specifications of the callback function that encrypts communication messages with custom encryption.

Format	IW_EXCRYPTO *CS_ExEncrypto(CS_REQ_DATA *req, char *buff, int buflen)
Argument	<p>req :Address of the area where request information is stored.</p> <p>buff :Address of the area where the message to be encrypted is stored.</p> <p>buflen :Message data length.</p>
Return value	<p>The function returns one of the following values.</p> <p>Other than NULL: Encryption successful or failed (with error message)</p> <p>NULL: Encryption failed (without error message)</p>
Description	<p>Perform the following in this callback function.</p> <ol style="list-style-type: none">(1) Abnormal termination if the req argument is NULL or invalid(2) Message encryption(3) Storage of encrypted message to the encryption structure <p>Operation of the Authentication Module changes as follows depending on the value set to each member of the encryption structure indicated by the return value.</p> <ul style="list-style-type: none">• $\text{buff} \neq \text{NULL}$ and $\text{buflen} \geq 1$ and $\text{result} = 0$ Encryption successful. <code>errmsg</code> is ignored.• Other than successful ($\text{buff} = \text{NULL}$ or $\text{buflen} < 1$ or $\text{result} \neq 0$) The Authentication Module judges that encryption has failed and outputs the result and <code>errmsg</code> values as messages to the error log.
Restrictions	<p>For the return value of this callback function, do not set a value other than the address of the encryption structure obtained by <code>IW_ExCPTCreateCrypto()</code> mentioned below or NULL.</p> <p>The argument of this callback function is for reference only. Do not change the value.</p>

CS_ExDecrypto

The following gives the specifications of the callback function that decrypts communication messages with custom encryption.

Format	IW_EXCRYPTO *CS_ExDecrypto(CS_REQ_DATA *req, char *buff, int buflen)
Argument	<p>req :Address of the area where request information is stored.</p> <p>buff :Address of the area where the message to decrypt is stored.</p> <p>buflen :Message data length.</p>
Return value	<p>The function returns one of the following values.</p> <p>Other than NULL: Decryption successful or failed (with error message)</p> <p>NULL: Decryption failed (without error message)</p>
Description	<p>Perform the following in this callback function.</p> <ol style="list-style-type: none">(1) Abnormal termination if the req argument is NULL or invalid(2) Message decryption(3) Storage of decrypted message to the encryption structure <p>Operation of the Authentication Module changes as follows depending on the value set to each member of the encryption structure indicated by the return value.</p> <ul style="list-style-type: none">• $\text{buff} \neq \text{NULL}$ and $\text{buflen} \geq 1$ and $\text{result} = 0$ Decryption successful. <code>errmsg</code> is ignored.• Other than successful ($\text{buff} = \text{NULL}$ or $\text{buflen} < 1$ or $\text{result} \neq 0$) The Authentication Module judges that decryption has failed and outputs the result and <code>errmsg</code> values as messages to the error log.
Restrictions	<p>For the return value of this callback function, do not set a value other than the address of the encryption structure obtained by <code>IW_ExCPTCreateCrypto()</code> mentioned below or NULL.</p> <p>The argument of this callback function is for reference only. Do not change the value.</p>

6.2 API function

The following API functions can be called within the callback functions.

API name	Process
IW_ExCPTCreateCrypto()	Creates the encryption structure.
IW_ExCPTReleaseCrypto()	Releases the encryption structure.
CS_ExGetIWEncryptedType()	Obtains the value of the X-iv-encrypted header included in the ICP 2.0 request.

The following pages describe the specifications of each API function.

IW_ExCPTCreateCrypto

The following gives the specifications of the API function for creating the encryption structure referenced in the return value of the callback function.

Format	IW_EXCRYPTO *IW_ExCPTCreateCrypto(char* buff, int len, int result, char* errmsg)
Argument	<p>buff :Specify the address of the area where the message to be encrypted (or decrypted) that is returned from the callback function is stored.</p> <p>len :Specify the data length of the message.</p> <p>result :Specify 0 to terminate the callback function as successful. Specify an error code to terminate it as failed. You can specify an arbitrary value other than "0" for the error code.</p> <p>errmsg :Specify the address of the area where the error message returned when the callback function is terminated as failed is stored.</p>
Return value	<p>The function returns one of the following values.</p> <p>Other than NULL: Address of the encryption structure</p> <p>NULL: Value could not be obtained due to an internal error</p>
Restrictions	<p>If the len argument value is less than the quantity of data stored in the buffer specified by the buff argument, data up to the length specified by len is stored.</p> <p>If the len argument value is negative, no data is stored even if data exists in the buffer specified by the buff argument.</p> <p>Even when the buff and len argument values are NULL and negative, respectively, if the result and/or errmsg arguments are specified, they are stored.</p> <p>If the errmsg argument value exceeds 200 characters, the error message may be truncated to 200 characters when the Authentication Module outputs it to the error log.</p> <p>When you do not use the area obtained by this API as the return value of the callback function, call IW_ExCPTReleaseCrypto() to destroy the area.</p>

IW_ExCPTReleaseCrypto

The following gives the specifications of the API function for releasing the encryption structure created by IW_ExCPTCreateCrypto().

Format	int IW_ExCPTReleaseCrypto (IW_EXCRYPTO* excrypto)
Argument	excrypto :Address of the encryption structure.
Return value	The function returns one of the following values. 0: Release successful Other than 0: Release failed
Restrictions	For the argument of this API, do not specify a value other than the address of the encryption structure obtained by IW_ExCPTCreateCrypto().

CS_ExGetIWEncryptedType

The following gives the specifications of the API function for obtaining the encryption method for custom encryption.

Format	char *CS_ExGetIWEncryptedType(CS_REQ_DATA *req)
Argument	req :Specify the address of the area where request information is stored.
Return value	The function returns one of the following values. Other than NULL: String in X-iw-encrypted header If the X-iw-encrypted header does not exist, "icewall" is returned. NULL: Value could not be obtained due to an internal error
Restrictions	This API is used only for referencing the X-iw-encrypted header value. Do not change the value.

7 Notes and Remarks on Implementing ICP

This chapter describes the notes and restrictions applied when you implement ICP 2.0.

7.1 Remarks

It is assumed that each message is encrypted for communication in ICP. The encryption algorithm used must be identical to that of the communication message encryption library used by the Authentication Module.

7.2 Restrictions

The specifications are subject to change without prior notice.

8 Notes and Restrictions on Developing the Communication Message Encryption Library

This chapter describes the notes and restrictions applied when you develop the communication message encryption library.

8.1 Remarks

(1) Scope of library

Because the process executed within the library is closely related to the internal processing of the Authentication Module, when a system error occurs within the library, the entire process may abort. Be sure to thoroughly test the developed library before integrating it into the Authentication Module.

Also, do not call `exit()` from the library because doing so terminates the running process.

(2) Performance degrading

If encryption or decryption imposes a heavy load on the Authentication Module, the entire performance of the module may be degraded.

(3) Memory leak

Be sure to release the memory area reserved by the library. Failure to do so may cause a memory leak.

(4) Support

Support for the product is available only when the standard library is used.

Support is not available if a custom encryption logic is used. Be sure to back up the standard library because it is necessary when you request for support.

8.2 Restrictions

Pay attention to the following restrictions when developing the encryption library. If you do not observe them during development, the Authentication Module may no longer be executable or its operation may become unstable.

(1) Build a 64-bit library.

The Authentication Module is designed to use 64-bit libraries.

You have to develop a 64-bit library to use with the 64-bit version of the Authentication Module.

(2) Use the thread-safe versions of the standard library functions and user functions.

The Authentication Module uses threads for the internal processing. The communication message encryption library is called from the Authentication Module as a function within a thread. This is why you need to use the thread-safe

versions of the standard library and user functions within the communication message encryption library.

- (3) Do not link different versions of the standard libraries within an archive.

The Authentication Module is designed to operate through dynamic use of libraries.

However, if an archive library is linked to the communication message encryption library, the library linked with the encryption library is used and a mismatch due to different library versions may occur. (A trouble that cannot be resolved by applying patches to the OS may occur.)

Note) In particular, you should take extra care about the state of the linked library when using a library of which organization (i.e., organization of libclntsh.sl) differs depending on the installation environment (e.g., Oracle) is used within the encryption library.

9 ICP 2.0 Client Sample

This chapter provides client sample source codes for using ICP 2.0. Two types of sample codes are provided respectively for C language and Java.

The specifications common to the sample codes are as follows:

- When login fails, no access control is performed.
- The message encryption/decryption algorithm used is the sample given in "10 Sample Communication Message Encryption Library."
- The user logs in using the user "ID=01" and "password=01."
- The Authentication Module configures the user group and access control of the login user.

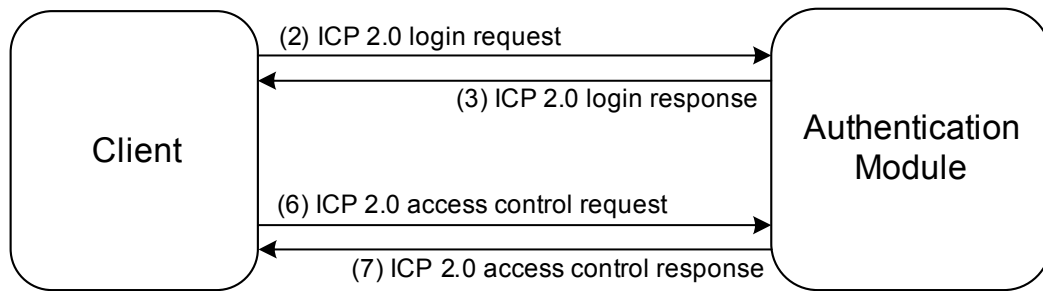
Note that by using the Java Agent Library (for ICP 2.0), you can communicate with the Authentication Module via ICP 2.0 without creating socket based Java code as the samples in this manual.

The sample source code consists of the following directories and files.

Directory			Description
/opt/icewall-ss0/sample_src/ICP20	/sample1	Makefile	Makefile
		Sample1.java	Client sample 1 (for Java)
		sample1.c	Client sample 1 (for C)
	/sample2	Makefile	Makefile
		Sample2.java	Client sample 2 (for Java)
		sample2.c	Client sample 2 (for C)
	/agent	Makefile	Makefile
		mod_sample.c	Agent sample (for C)

9.1 Client sample 1

This sample sends login and access control requests from the client to the Authentication Module using ICP 2.0.

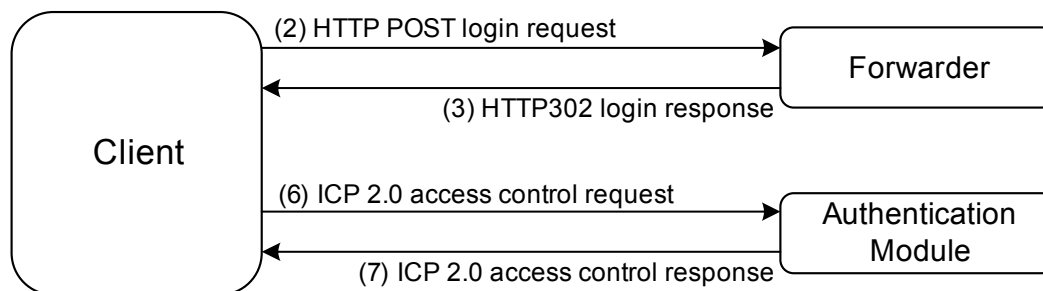


This sample performs the following:

- (1) Creates a login request message.
- (2) Sends the login request to the Authentication Module using ICP 2.0.
- (3) Receives a login response from the Authentication Module using ICP 2.0.
- (4) Obtains the session ID from the login response.
- (5) Creates an access control request message using the obtained session ID.
- (6) Sends the access control request to the Authentication Module using ICP 2.0.
- (7) Receives an access control response from the Authentication Module using ICP 2.0.
- (8) Obtains the access control result from the access control response.

9.2 Client sample 2

This sample logs in to the Forwarder from the client via HTTP communication, and then sends an access control request to the Authentication Module using ICP 2.0.



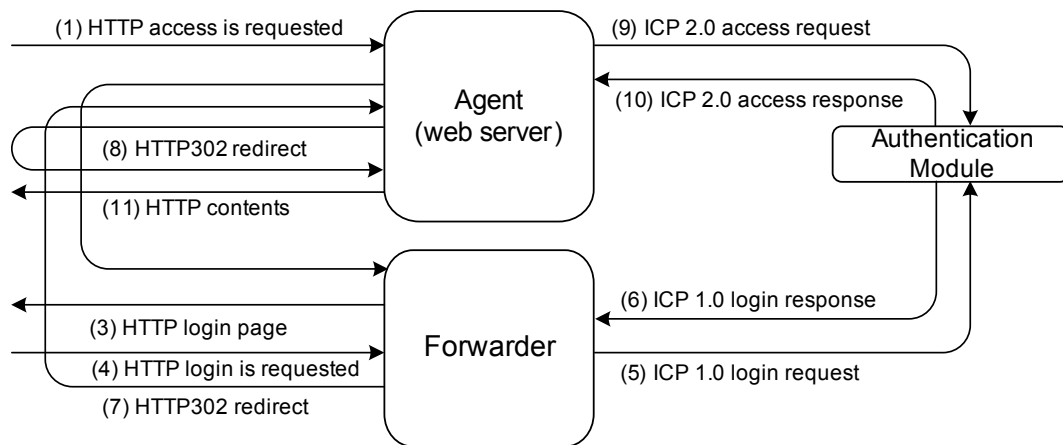
This sample performs the following:

- (1) Creates a login request message using a user ID and password.
- (2) Sends the login request to the Forwarder via HTTP using the POST method.
- (3) Receives a login response from the Forwarder.
- (4) Obtains the session ID from the login response.
- (5) Creates an access control request message to be sent to the Authentication Module using the obtained session ID.
- (6) Sends the access control request to the Authentication Module using ICP 2.0.

- (7) Receives an access control response from the Authentication Module using ICP 2.0.
- (8) Obtains the access control result.

9.3 Agent sample (for Apache HTTP Server 1.3.x)

This sample sends an access control request from an agent built on the Web server to the Authentication Module using ICP 2.0. As with the Agent Option, the login request is processed by the Forwarder and then control is returned to the agent.



This sample performs the following:

- (1) Sends an access request from the client to the agent (Web server) via HTTP.
- (2) The agent judges if the user has already logged in to the IceWall, and redirects the user to the Forwarder if he or she has not logged in.
- (3) The Forwarder sends the Login page to the client.
- (4) Sends a login request for the IceWall with the user ID and password entered from the client to the Forwarder.
- (5) The Forwarder sends the login request to the Authentication Module using ICP 1.0.
- (6) The Forwarder receives the login response from the Authentication Module using ICP 1.0.
- (7) If the login is successful, the Forwarder redirects the result to the agent and sends the session ID to the client.
- (8) The agent redirects the result to the client to notify the client of the session ID sent from the Forwarder. The redirect target is the URL specified in (1).
- (9) The agent sends the access control request to the Authentication Module using ICP 2.0.
- (10) The agent receives an access control response from the Authentication Module using ICP 2.0.
- (11) If the access control is successful, the agent sends the requested contents to the client.

This process uses the same sequence as used by the Agent Option. Therefore, you need to send necessary information in the specified format. The format is shown below.

Process number	Format
(2)	HTTP/1.1 302 Moved Temporary Location: [<i>forwarder URL</i>]?[<i>identification keyword</i>]=[<i>server name in requested URL</i>][<i>identification dummy path</i>]&path=[<i>path in requested URL</i>]&query=[<i>requested argument</i>]
(7)	HTTP/1.1 302 Moved Temporary Location: [<i>server name in requested URL</i>][<i>identification dummy path</i>]?[<i>identification keyword</i>]=[<i>session ID</i>]&path=[<i>path in requested URL</i>]&query=[<i>requested argument</i>]
(8)	HTTP/1.1 302 Moved Temporary Location: [<i>requested URL</i>]? [<i>requested argument</i>] Set-Cookie: [<i>authentication cookie name</i>]=[<i>session ID</i>]

Each parameter (in italic) is obtained as follows.

Parameter name	Obtained from
<i>Forwarder URL</i>	DFW_PATH in the Web server configuration file (httpd.conf).
<i>Identification keyword</i>	Fixed to "AGENT_KEY."
<i>Requested URL</i>	URL specified by the client in (1).
<i>Requested argument</i>	Argument in the URL specified by the client in (1).
<i>Identification dummy path</i>	Fixed to "/sample."
<i>Authentication cookie name</i>	Fixed to "IW_INFO."

The specifications of this agent sample are as follows:

- The Authentication Module information for connection is fixed.
- Each parameter is URL-encoded upon redirection.
- Only the user ID and session ID are added as the HTTP header.
- If an error occurs or access control fails, the process ends with Status 403.
- Logs are not output.
- This sample does not run on the same Web server as the IceWall server.

You need to add the following to the Web server configuration file (httpd.conf) to use this agent sample.

```
~
LoadModule sample_module /opt/icewall-ss0/sample_src/ICP20/agent/mod_sample.so
AddModule mod_sample.c
~
<VirtualHost _default_:80>
~
  <Location />
    DFW_PATH http://dfw_host/fw/dfw
  ErrorDocument 403 "mod_sample error"
  </Location>
~
</VirtualHost>
```

Specify the full path to the sample agent library on the LoadModule line.
Specify the source file that contains the main function of the module on the AddModule line.
Specify the URL of the Forwarder that processes login on the DFW_PATH line.
After configuring these settings, restart the Web server.

9.4 Sample source (C language)

The sample source written in C language is shown below.

9.4.1 Client sample 1 (sample1.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/socket.h>

#define LOGIN_MODE 1
#define ACCESS_MODE 2
#define CRYPTOKEY "yGHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJXEAXx7hGQgY"

int CreateRequest( int mode, char *request, char *sessionid );
void CheckResponse( int mode, char *response, char *data );
void SocketCertd( int sendsize, char *request, char *response );
void ChangeEOF( char *buff, int len );

/*-----*/
int main( int argc, char **argv )
{
```



```
char request[4096];
char response[4096];
char sessionid[33];
char errcode[3];
int sendsize;

/* Login */
sendsize = CreateRequest( LOGIN_MODE, request, NULL );
SocketCertd( sendsize, request, response );
CheckResponse( LOGIN_MODE, response, sessionid );
/* Login Error */
if( strlen( sessionid ) == 0 ) exit( -1 );

/* Access */
sendsize = CreateRequest( ACCESS_MODE, request, sessionid );
SocketCertd( sendsize, request, response );
CheckResponse( ACCESS_MODE, response, errcode );

exit( atoi( errcode ) );
}

void SocketCertd( int sendsize, char *request, char *response )
{
    int      sock;
    struct sockaddr_in addr;
    struct hostent  *hent;

    /* Certd ( localhost:14142 ) */
    memset( &addr, 0x00, sizeof(struct sockaddr_in) );
    addr.sin_family = AF_INET;
    addr.sin_port = htons( 14142 );
    hent = gethostbyname( "localhost" );
    memcpy( &addr.sin_addr, hent->h_addr, hent->h_length );

    sock = socket( AF_INET, SOCK_STREAM, 0 );

    connect( sock, (struct sockaddr*)&addr, sizeof( addr ) );

    send( sock, request, sendsize, 0 );

    recv( sock, response, 4096, 0 );

    shutdown( sock, SHUT_RDWR );
    close( sock );
}

int CreateRequest( int mode, char *request, char *sessionid )
{
    /* Messeage -> Encrypto */
    switch( mode ) {
    case LOGIN_MODE:
        strcpy( request,
```

```
        "REQ / ICP/2.0\r\nContent-length: 43\r\n\r\nMSG_ID: ReqLoginUID\r\nIW_UID: 01\r\nIW_PWD: 01");
        ChangeEOF( ( request + 37 ), 43 );
        break;

    case ACCESS_MODE:
        sprintf( request,
            "REQ / ICP/2.0\r\nContent-length: 78\r\n\r\nMSG_ID: ReqAccessUID\r\nIW_INFO: %s\r\nACC_URL: http",
                sessionid );
        ChangeEOF( ( request + 37 ), 78 );
        break;
    }

    return( strlen( request ) );
}

void CheckResponse( int mode, char *response, char *data )
{
    char    *ptr;
    char    *buff;
    int     len;

    /* Decrypt */
    buff = strstr( response, "\r\n\r\n" ) + 4;
    len = atoi( strstr( response, "Content-length: " ) + 16 );
    ChangeEOF( buff, len );

    switch( mode ) {
    case LOGIN_MODE:
        /* Get SessionID */
        ptr = strstr( buff, "IW_INFO: " ) + 9;
        if( ptr != NULL ) strncpy( data, ptr, 32 );
        break;

    case ACCESS_MODE:
        /* Get ErrNo */
        ptr = strstr( buff, "ERR_NO: " ) + 8;
        if( ptr != NULL ) strncpy( data, ptr, 2 );
        break;
    }
}

void ChangeEOF( char *buff, int len )
{
    char ascii;
    int count;
    int index;

    index = 0;
    for( count = 0; count < len; count++) {
```

```
/* ASCII 4bit */
ascii = CRYPTOKEY[index++] & 0x0F;
if( index >= 54 ) index = 0;

/* EOR */
buff[count] ^= ascii;

/* 0x7f */
if( buff[count] == 0x7F ) buff[count] ^= ascii;
}
}
```

9.4.2 Client sample 2 (sample2.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/socket.h>

#define LOGIN_MODE 1
#define ACCESS_MODE 2
#define CRYPTOKEY "ygHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJXEAXx7hGQgY"

int CreateRequest( int mode, char *request, char *sessionid );
void CheckResponse( int mode, char *response, char *data );
void SocketMessage( int port, int sendsize, char *request, char *response );
void ChangeEOF( char *buff, int len );

/*-----*/
int main( int argc, char **argv )
{
    char request[4096];
    char response[4096];
    char sessionid[33];
    char errcode[3];
    int sendsize;

    /* Login : dfw(localhost:80) */
    sendsize = CreateRequest( LOGIN_MODE, request, NULL );
    SocketMessage( 80, sendsize, request, response );
    CheckResponse( LOGIN_MODE, response, sessionid );

    /* Login Error */
    if( strlen( sessionid ) == 0 ) exit( -1 );
}
```

```
/* Access : certd(localhost:14142) */
sendsize = CreateRequest( ACCESS_MODE, request, sessionid );
SocketMessage( 14142, sendsize, request, response );
CheckResponse( ACCESS_MODE, response, errcode );

exit( atoi( errcode ) );
}

void SocketMessage( int port, int sendsize, char *request, char *response )
{
    int          sock;
    struct sockaddr_in addr;
    struct hostent  *hent;

    memset( &addr, 0x00, sizeof(struct sockaddr_in) );
    addr.sin_family = AF_INET;
    addr.sin_port   = htons( port );
    hent = gethostbyname( "localhost" );
    memcpy( &addr.sin_addr, hent->h_addr, hent->h_length );

    sock = socket( AF_INET, SOCK_STREAM, 0 );

    connect( sock, (struct sockaddr*)&addr, sizeof( addr ) );

    send( sock, request, sendsize, 0 );

    recv( sock, response, 4096, 0 );

    shutdown( sock, SHUT_RDWR );
    close( sock );
}

int CreateRequest( int mode, char *request, char *sessionid )
{
    switch( mode ) {
        case LOGIN_MODE:
            /* Login postData */
            strcpy( request,
                "POST /fw/dfw HTTP/1.0\r\nContent-length: 84\r\nHost: localhost:80\r\n\r\nACCOUNTUID=01&PASSWORD=01&HIDEURL=%2Fhttp%2F&LOGIN=ICEWALL_LOGIN&SUBMIT=+%91%97%90M+" );
            break;

        case ACCESS_MODE:
            /* Icp20 Access */
            sprintf( request,
                "REQ / ICP/2.0\r\nContent-length: 78\r\n\r\nMSG_ID: ReqAccessUID\r\nIW_INFO: %s\r\nACC_URL: http",
                sessionid );

            /* Encrypto */
    }
```

```
        ChangeEOF( ( request + 37 ), 78 );
        break;
    }

    return( strlen( request ) );
}

void CheckResponse( int mode, char *response, char *data )
{
    char    *ptr;
    char    *buff;
    int     len;
    switch( mode ) {
    case LOGIN_MODE:
        /* Get SessionID */
        ptr = strstr( response, "Set-Cookie: IW_INFO=" ) + 20;
        if( ptr != NULL ) strncpy( data, ptr, 32 );
        break;

    case ACCESS_MODE:
        /* Decrypt */
        buff = strstr( response, "\r\n\r\n" ) + 4;
        len = atoi( strstr( response, "Content-length: " ) + 16 );
        ChangeEOF( buff, len );

        /* Get ErrNo */
        ptr = strstr( buff, "ERR_NO: " ) + 8;
        if( ptr != NULL ) strncpy( data, ptr, 2 );
        break;
    }
}

void ChangeEOF( char *buff, int len )
{
    char ascii;
    int count;
    int index;

    index = 0;
    for( count = 0; count < len; count++ ){

        /* ASCII 4bit */
        ascii = CRYPTOKEY[index++] & 0x0F;
        if( index >= 54 ) index = 0;

        /* EOR */
        buff[count] ^= ascii;

        /* 0x7f */
        if( buff[count] == 0x7F ) buff[count] ^= ascii;
    }
}
```

9.5 Sample source (Java language)

The sample source written in Java language is shown below.

9.5.1 Client sample 1 (Sample1.java)

```
import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.StringTokenizer;

/**
 * SampleA
 *
 */
public class SampleA {

    private final String CRYPTOKEY = "ygHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJXEAXx7hGQgY";
    private final int LOGIN_MODE = 0;
    private final int ACCESS_MODE = 1;

    /** certd host name */
    private String i_host = "localhost";
    /** certd port number */
    private int i_port = 14142;
    /** User ID */
    private String i_uid = "user01";
    /** Password*/
    private String i_pwd = "user01";

    /**
     * Constructor.
     *
     */
    SampleA(String[] args) {

        //Login process
        String loginRequest = createRequest(LOGIN_MODE, null);
        String response = socketCertd(loginRequest);
        String sessionId = checkResponse(LOGIN_MODE, response);

        //Login error
        if(sessionId == null || sessionId.length() == 0)
            System.exit(-1);

        //Access
        String accessRequest = createRequest(ACCESS_MODE, sessionId);
        response = socketCertd(accessRequest);
        String errCode = checkResponse(ACCESS_MODE, response);
    }
}
```

```
try{
    System.exit( Integer.parseInt(errCode) );
}catch(NumberFormatException e){
    System.exit(-1);
}
}

/**
 * Connects to certd, sends the request, and returns the result.
 *
 * @param request Send request
 * @return Response. NULL if it fails
 */
private String socketCertd(String request) {

    String resData = null;
    try {
        Socket socket = new Socket( i_host, i_port );

        // Sends message
        DataOutputStream out = new DataOutputStream(socket.getOutputStream());
        out.write(request.getBytes());
        out.flush();

        // Receives message
        byte[] recvbuff = new byte[0];
        int buffsize = 0;
        byte[] workbuff = new byte[1024];

        DataInputStream in = new DataInputStream(socket.getInputStream());
        while(true) {
            int recvsized = in.read(workbuff, 0, workbuff.length);
            if (recvsized == -1) {
                break;
            }

            if (buffsize == 0) {
                recvbuff = new byte[recvsized];
                System.arraycopy(workbuff, 0, recvbuff, 0, recvsized);
                buffsize = recvsized;
            } else {
                byte[] tempbuff = new byte[buffsize + recvsized];
                System.arraycopy(recvbuff, 0, tempbuff, 0, buffsize);
                System.arraycopy(workbuff, 0, tempbuff, buffsize, recvsized);
                recvbuff = tempbuff;
                buffsize = buffsize + recvsized;
            }
        }
        resData = new String(recvbuff);

        out.close();
        in.close();
    }
}
```

```
        socket.close();
    }
    catch( ArrayIndexOutOfBoundsException e ) {
        System.err.println("Usage:java MessageClient hostname");
        System.exit(-1);
    }
    catch( UnknownHostException e ) {
        System.err.println( "Host not found." );
        System.exit(-1);
    }
    catch( SocketException e ) {
        System.err.println("Socket Error!" + e);
        System.exit(-1);
    }
    catch( IOException e ) {
        System.err.println("IO Error!");
        System.exit(-1);
    }
    return resData;
}

/**
 * Generates a request string.
 *
 * @param mode Mode
 * @param sessionId Session ID
 * @return Request string
 */
private String createRequest(int mode, String sessionId) {
    String request = null;
    StringBuffer header = new StringBuffer();
    StringBuffer body = new StringBuffer();
    switch(mode){
    case LOGIN_MODE:
        body.append("MSG_ID: ReqLoginUID\r\n");
        body.append("IW_UID: " + i_uid + "\r\n");
        body.append("IW_PWD: " + i_pwd);
        header.append("REQ / ICP/2.0\r\n");
        header.append("Content-length: " + body.length() + "\r\n");
        header.append("\r\n");
        request = header.toString() + changeEOF(body.toString());
        break;
    case ACCESS_MODE:
        body.append("MSG_ID: ReqAccessUID\r\n");
        body.append("IW_INFO: " + sessionId + "\r\n");
        body.append("ACC_URL: http");
        header.append("REQ / ICP/2.0\r\n");
        header.append("Content-length: " + body.length() + "\r\n");
        header.append("\r\n");
        request = header.toString() + changeEOF(body.toString());
        break;
    }
}
```



```
    return request;
}

/**
 * Checks the response.
 *
 * @param mode Mode
 * @param resData Response
 * @return String obtained by analyzing the response
 */
private String checkResponse(int mode, String resData) {
    String result = null;
    if(resData == null || resData.length() == 0)
        return null;

    // Extracts header
    int wkIdx = 0;
    String resLine = resData.substring(wkIdx, resData.indexOf("\r\n"));
    wkIdx += resLine.length() + 2;
    String dataLenLine = resData.substring(wkIdx, resData.indexOf("\r\n", wkIdx
));
    // Extracts body
    wkIdx += dataLenLine.length() + 4;
    String body = changeEOF(resData.substring(wkIdx));

    String key = null;
    switch(mode){
    case LOGIN_MODE:
        key = "TW_INFO: ";
        if(body.indexOf(key) >= 0){
            int idx = body.indexOf(key) + key.length();
            result = body.substring(idx, idx + 32);
        }
        break;
    case ACCESS_MODE:
        StringTokenizer st = new StringTokenizer(body, "\r\n");
        String line = null;
        key = "ERR_NO: ";
        for( ; st.hasMoreElements(); ) {
            line = st.nextToken();
            if(line.indexOf(key) >= 0){
                result = line.substring(line.indexOf(key) + key.length());
                break;
            }
        }
        break;
    default:
        return null;
    }

    return result;
}
```

```
/**
 * EOF conversion of string.
 *
 * @param base Source string
 * @return Converted string
 */
private String changeEOF(String base) {
    byte[] buff = base.getBytes();
    int len = buff.length;
    byte ascii;
    int count;
    int index;

    index = 0;
    for( count = 0; count < len; count++){

        /* ASCII 4bit */
        ascii = (byte)((int)CRYPTOKEY.charAt(index) & 0x0F);
        index++;
        if( index >= 54 ) index = 0;

        /* EOR */
        buff[count] ^= ascii;

        /* 0x7f */
        if( buff[count] == 0x7F ) buff[count] ^= ascii;

    }
    return new String(buff);
}

public static void main(String[] args) {
    new SampleA(args);
}
}
```

9.5.2 Client sample 2 (sample2.java)

```
import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.StringTokenizer;

/**
 * SampleB
 *
 */
public class SampleB {
```

```
private final String CRYPTOKEY = "ygHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJXEAXx7hGQgY";
private final int LOGIN_MODE = 0;
private final int ACCESS_MODE = 1;

/** dfw host name */
private String i_dfwHost = "localhost";
/** dfw port number */
private int i_dfwPort = 80;
/** certd host name */
private String i_certdHost = "localhost";
/** certd port number */
private int i_certdPort = 14142;
/** User ID */
private String i_uid = "user01";
/** Password*/
private String i_pwd = "user01";

/**
 * Constructor.
 */
SampleB(String[] args) {

    if(args.length > 0)
        i_dfwHost = args[0];
    if(args.length > 1)
        i_dfwPort = Integer.parseInt(args[1]);
    if(args.length > 2)
        i_certdHost = args[2];
    if(args.length > 3)
        i_certdPort = Integer.parseInt(args[3]);

    //Login process
    String loginRequest = createRequest(LOGIN_MODE, null);
    String response = socketMessage(i_dfwHost, i_dfwPort, loginRequest);
    String sessionId = checkResponse(LOGIN_MODE, response);

    //Login error
    if(sessionId == null || sessionId.length() == 0)
        System.exit(-1);

    //Access
    String accessRequest = createRequest(ACCESS_MODE, sessionId);
    response = socketMessage(i_certdHost, i_certdPort, accessRequest);
    String errCode = checkResponse(ACCESS_MODE, response);
    try{
        System.exit( Integer.parseInt(errCode) );
    }catch(NumberFormatException e){
        System.exit(-1);
    }
}
```

```
}

/**
 * Performs socket communication. Sends the request and returns the result.
 *
 * @param host Connection target host name
 * @param port Connection target port
 * @param request Send request
 * @return Response. NULL if it fails
 */
private String socketMessage(String host, int port, String request) {
    String resData = null;
    try {
        Socket socket = new Socket( host, port );

        // Sends message
        DataOutputStream out = new DataOutputStream(socket.getOutputStream
());
        out.write(request.getBytes());
        out.flush();

        // Receives message
        byte[] recvbuff = new byte[0];
        int buffsize = 0;
        byte[] workbuff = new byte[1024];

        DataInputStream in = new DataInputStream(socket.getInputStream());
        while(true) {
            int recvsized = in.read(workbuff, 0, workbuff.length);
            if (recvsized == -1) {
                break;
            }

            if (buffsize == 0) {
                recvbuff = new byte[recvsized];
                System.arraycopy(workbuff, 0, recvbuff, 0, recvsized);
                buffsize = recvsized;
            } else {
                byte[] tempbuff = new byte[buffsize + recvsized];
                System.arraycopy(recvbuff, 0, tempbuff, 0, buffsize);
                System.arraycopy(workbuff, 0, tempbuff, buffsize, recvsized);
                recvbuff = tempbuff;
                buffsize = buffsize + recvsized;
            }
        }
        resData = new String(recvbuff);

        out.close();
        in.close();
        socket.close();
    }
    catch( ArrayIndexOutOfBoundsException e ) {
```

```
        System.err.println("Usage:java MessageClient hostname");
        System.exit(-1);
    }
    catch( UnknownHostException e ) {
        System.err.println( "Host not found." );
        System.exit(-1);
    }
    catch( SocketException e ) {
        System.err.println("Socket Error!");
        System.exit(-1);
    }
    catch( IOException e ) {
        System.err.println("IO Error!");
        System.exit(-1);
    }
    return resData;
}

/**
 * Generates a request string.
 *
 * @param mode Mode
 * @param sessionId Session ID
 * @return Request string
 */
private String createRequest(int mode, String sessionId) {
    String request = null;
    StringBuffer header = new StringBuffer();
    StringBuffer body = new StringBuffer();
    switch(mode){
        case LOGIN_MODE:
            body.append("ACCOUNTUID=");
            body.append(i_uid);
            body.append("&PASSWORD=");
            body.append(i_pwd);

body.append("&HIDEURL=%2Fhttp&LOGIN=ICEWALL_LOGIN&SUBMIT=+%9
1%97%90M+");
            header.append("POST /fw70sp2/dfw HTTP/1.0\r\n");
            header.append("Content-length: " + body.length() + "\r\n");
            header.append("Host: ");
            header.append(i_dfwHost);
            header.append(":");
            header.append(i_dfwPort);
            header.append("\r\n\r\n");
            request = header.toString() + body.toString();
            break;
        case ACCESS_MODE:
            body.append("MSG_ID: ReqAccessUID\r\n");
            body.append("IW_INFO: " + sessionId + "\r\n");
            body.append("ACC_URL: http");
            header.append("REQ / ICP/2.0\r\n");
    }
}
```

```
        header.append("Content-length: " + body.length() + "\r\n");
        header.append("\r\n");
        request = header.toString() + changeEOF(body.toString());
        break;
    }
    return request;
}

/**
 * Checks the response.
 *
 * @param mode Mode
 * @param resData Response
 * @return String obtained by analyzing the response
 */
private String checkResponse(int mode, String resData) {
    String result = null;
    if(resData == null || resData.length() == 0)
        return null;

    String key = null;
    switch(mode){
    case LOGIN_MODE:
        key = "Set-Cookie: IW_INFO=";
        if(resData.indexOf(key) >= 0){
            int idx = resData.indexOf(key)+key.length();
            result = resData.substring(idx, idx+32);
        }
        break;
    case ACCESS_MODE:
        // Extracts header
        int wkIdx = 0;
        String resLine = resData.substring(wkIdx, resData.indexOf("\r\n"));
        wkIdx += resLine.length() + 2;
        String dataLenLine = resData.substring(wkIdx, resData.indexOf("\r\n", wkI
dx));
        // Extracts body
        wkIdx += dataLenLine.length() + 4;
        String body = changeEOF(resData.substring(wkIdx));
        StringTokenizer st = new StringTokenizer(body, "\r\n");
        String line = null;
        key = "ERR_NO: ";
        for( ; st.hasMoreElements(); ) {
            line = st.nextToken();
            if(line.indexOf(key) >= 0){
                result = line.substring(line.indexOf(key) + key.length());
                break;
            }
        }
        break;
    default:
        return null;
    }
}
```

```
    }

    return result;
}

/**
 * EOF conversion of string.
 *
 * @param base Source string
 * @return Converted string
 */
private String changeEOF(String base) {
    byte[] buff = base.getBytes();
    int len = buff.length;
    byte ascii;
    int count;
    int index;

    index = 0;
    for( count = 0; count < len; count++){

        /* ASCII 4bit */
        ascii = (byte)((int)CRYPTOKEY.charAt(index) & 0x0F);
        index++;
        if( index >= 54 ) index = 0;

        /* EOR */
        buff[count] ^= ascii;

        /* 0x7f */
        if( buff[count] == 0x7F ) buff[count] ^= ascii;

    }
    return new String(buff);
}

public static void main(String[] args) {
    new SampleB(args);
}
}
```

9.6 Sample source (Agent sample: mod_sample.c)

The agent sample (written in C language) is provided for Apache1.3 only.

```
#include "httpd.h"
#include "http_config.h"
#include "http_protocol.h"
#include "ap_config.h"

/* define */
#define CERT          "localhost"
#define PORT          14142
#define AGENT_KEY      "AGENT_KEY"
#define AGENT_PATH     "/sample"
#define COOKIENAME     "IW_INFO"
#define AGENT_KEY_EQ   "AGENT_KEY="
#define COOKIENAME_EQ  "IW_INFO="
#define CRYPTOKEY      "ygHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJXEAXx7hGQgY"

module MODULE_VAR_EXPORT sample_module;

typedef struct{
    char *dfw_path;
} sample_dir_config;

/* prototype */
char *EncodeData( request_rec *r, char *old_buf );
void LoginRedirect( request_rec *r, sample_dir_config *conf, char *path, char *query );
void SetCookieRedirect( request_rec *r, char *query );
char *CreateRequest( request_rec *r, char *path, char *cookie );
int  GetErrNo( request_rec *r, char *res_body );
void SetHTTPHeader( request_rec *r, char *res_body, char *cookie );
void SocketCertd( int sendsize, char *request, char *response );
void ChangeEOF( char *buff, int len );

/* check access by host address */
/*-----*/
/*    Main process                                */
/*                                */
/*    1. Obtains the pointer to the httpd.conf setting value structure. */
/*    2. Obtains different types of information.                        */
/*    3. Analyzes the request.                                          */
/*    4. Changes the status according to the request and returns it to the server. */
/*                                */
/*                                */
/*-----*/
static int SAMPLE_DFW_Main( r )
request_rec *r; /* Apache request structure pointer */
{
    sample_dir_config *conf; /* httpd.conf setting value structure pointer */
```



```
char *w_cookie;      /* Cookie */
char *w_path;        /* Requested URL */
char *w_query;       /* Requested QUERY_STRING */
char *request;       /* ICP2.0 request message */
char *response;      /* ICP2.0 response message */
char *res_body;      /* ICP2.0 response body section */
int w_err_no;        /* ICP2.0 response err_no (response code) */
int w_ret = DECLINED; /* Return value */
int sendsize;        /* Transfer size */

/* Obtains information. */
conf = ap_get_module_config( r->per_dir_config, &sample_module );
w_cookie = (char *)ap_table_get( r->headers_in, "COOKIE" );
w_path = r->uri;
w_query = r->args;

/* Checks for cookie. */
if( w_cookie == NULL ){
    /* Analyzes the request. */
    if( ( strcmp( w_path, AGENT_PATH, strlen( AGENT_PATH ) ) == 0 ) &&
        ( w_query != NULL && strcmp( w_query, AGENT_KEY_EQ, strlen(
AGENT_KEY_EQ ) ) == 0 ) ){
        /* Redirect for Set-Cookie */
        SetCookieRedirect( r, w_query );
    }else{
        /* Redirect for Login */
        LoginRedirect( r, conf, w_path, w_query );
    }
    w_ret = HTTP_MOVED_TEMPORARILY;
}else{
    /* Checks for cookie of IceWall. */
    if( ( w_cookie = strstr( w_cookie, COOKIENAME_EQ ) ) != NULL ){
        /* Analyzes the request. */
        if( ( strcmp( w_path, AGENT_PATH, strlen( AGENT_PATH ) ) == 0 ) &&
            ( w_query != NULL && strcmp( w_query, AGENT_KEY_EQ, strlen(
AGENT_KEY_EQ ) ) == 0 ) ){
            /* Redirect for Set-Cookie */
            SetCookieRedirect( r, w_query );
            w_ret = HTTP_MOVED_TEMPORARILY;
        }else{
            /* Access control process */
            /* Creates the ICP2.0 access control request message. */
            request = CreateRequest( r, w_path, w_cookie );

            /* Access control communication */
            sendsize = strlen( request );
            response = ap_palloc( r->pool, 4096 );
            memset( response, '\0', 4096 );
            SocketCertd( sendsize, request, response );

            /* Receives the ICP2.0 access control response message. */
            res_body = strstr( response, "\r\n\r\n" ) + 4;
```



```

char *w_en_chr; /* Temporary saving area for encoded characters */

if( old_buf == NULL || strlen( old_buf ) == 0 ){
    return( old_buf );
}

w_new_ptr = ap_pstrdup( r->pool, "" );
for ( w_ptr = old_buf; *w_ptr != '\0'; w_ptr++ ){
    switch( *w_ptr ){
        case ' ':
        case '"':
        case '#':
        case '%':
        case '{':
        case '}':
        case '|':
        case '\\':
        case '^':
        case '~':
        case '[':
        case ']':
        case '\':
        case ';':
        case '?':
        case ':':
        case '@':
        case '=':
        case '&':
            w_en_chr = ap_psprintf( r->pool, "%%%02x", *w_ptr );
            w_new_ptr = ap_pstrcat( r->pool, w_new_ptr, w_en_chr, NULL );
            break;
        default:
            w_en_chr = ap_psprintf( r->pool, "%c", *w_ptr );
            w_new_ptr = ap_pstrcat( r->pool, w_new_ptr, w_en_chr, NULL );
            break;
    }
}

return( w_new_ptr );
}

/*-----*/
/* Login redirect */
/* */
/* 1. Obtains the HTTP_HOST value. */
/* 2. URL-encodes each parameter set for QUERY_STRING. */
/* 3. Creates the Location header for redirect to DFW. */
/* 4. Sets the Location header to the Apache request structure. */
/* */
/*-----*/
void LoginRedirect( r, conf, path, query )
request_rec *r; /* Apache request structure pointer */

```

```
sample_dir_config *conf; /* httpd.conf setting value structure pointer */
char      *path; /* &path= portion */
char      *query; /* &query= portion */
{
    char *w_host; /* HTTP_HOST value */
    char *w_content_url; /* Content URL */
    char *w_en_content_url; /* Encoded content URL */
    char *w_en_path; /* Encoded &path= portion */
    char *w_en_content_query; /* Encoded &query= portion */
    char *w_redirect; /* String set to Location header */

    /* Redirect for Login */
    w_host = (char *)ap_table_get( r->headers_in, "HOST" );
    w_content_url = ap_psprintf( r->pool,
                                "%s://%s%s",
                                ap_http_method( r ),
                                w_host,
                                AGENT_PATH );

    /* Encode */
    w_en_content_url = EncodeData( r, w_content_url );
    w_en_path = EncodeData( r, path );

    /* Location Header */
    if( query == NULL || strlen( query ) == 0 ){
        w_redirect = ap_psprintf( r->pool,
                                "%s?%s=%s&path=%s&query=",
                                conf->dfw_path,
                                AGENT_KEY,
                                w_en_content_url,
                                w_en_path );
    }else{
        w_en_content_query = EncodeData( r, query );
        w_redirect = ap_psprintf( r->pool,
                                "%s?%s=%s&path=%s&query=%s",
                                conf->dfw_path,
                                AGENT_KEY,
                                w_en_content_url,
                                w_en_path,
                                w_en_content_query );
    }
    ap_table_set( r->err_headers_out, "Location", w_redirect );

    return;
}

/*-----*/
/* Set-Cookie redirect */
/* */
/* 1. Obtains the HTTP_HOST value. */
/* 2. URL-decodes information obtained from QUERY_STRING. */
/* 3. Creates the Location header for Set-Cookie redirect. */
/* 4. Sets the Location header to the Apache request structure. */
```

```

/* 5. Obtains the session information. */
/* 6. Creates the Set-Cookie header according to the session information. */
/* 7. Sets the Set-Cookie header to the Apache request structure. */
/*-----*/
void SetCookieRedirect( r, query )
request_rec *r; /* Apache request structure pointer */
char *query; /* QUERY_STRING */
{
    char *w_host; /* HTTP_HOST value */
    char *w_path; /* &path= portion */
    char *w_content_url; /* Content URL */
    char *w_content_query; /* Content QUERY_STRING */
    char *w_redirect; /* String set to Location header */
    char *w_session; /* Session information */
    char *w_query; /* QUERY_STRING */
    char *w_setcookie; /* String set to Set-Cookie header */

    /* Redirect for Set-Cookie */
    /* host */
    w_host = (char *)ap_table_get( r->headers_in, "HOST" );
    /* path */
    w_path = strstr( query, "path=" );
    if( w_path != NULL ){
        w_path = w_path + 5;
        w_path = ap_getword( r->pool, (const char **)&w_path, '&' );
        ap_unescape_url( w_path );
    }

    /* URL */
    w_content_url = ap_psprintf( r->pool,
                                "%s://%s%s",
                                ap_http_method( r ),
                                w_host,
                                w_path );

    /* query */
    w_content_query = strstr( query, "query=" );
    ap_unescape_url( w_content_query );

    /* Location Header */
    if( w_content_query != NULL && strlen( w_content_query ) > 6 ){
        w_content_query = w_content_query + 6;
        w_redirect = ap_psprintf( r->pool,
                                "%s?%s",
                                w_content_url,
                                w_content_query );
    }else{
        w_redirect = ap_psprintf( r->pool,
                                "%s",
                                w_content_url );
    }
}

```

```

    ap_table_set( r->err_headers_out, "Location", w_redirect );

    /* SessionID */
    w_query = ap_pstrdup( r->pool, query + strlen( AGENT_KEY_EQ ) );
    w_session = ap_getword( r->pool, (const char **)&w_query, '&' );

    /* Set-Cookie Header */
    w_setcookie = ap_psprintf( r->pool, "%s=%s; path=/", COOKIENAME, w_session );
    ap_table_set( r->err_headers_out, "Set-Cookie", w_setcookie );

    return;
}

/*-----*/
/*   Access control request message creation   */
/*                                           */
/*   1. Obtains the HTTP_HOST value.           */
/*   2. Creates the URL for access control.      */
/*   3. Obtains the session information.         */
/*   4. Creates and encrypts the body of the request message. */
/*   5. Creates and encrypts the header of the request message. */
/*   6. Creates the request message by combining the header and body. */
/*   7. Returns the request message (with the encrypted body). */
/*                                           */
/*-----*/
char *CreateRequest( r, path, cookie )
request_rec *r; /* Apache request structure pointer */
char *path; /* Access control target path */
char *cookie; /* Cookie for IceWall */
{
    char *w_host; /* HTTP_HOST value */
    char *w_content_url; /* Content URL */
    char *w_session; /* Session information */
    char *w_request; /* ICP2.0 access control request message */
    char *w_req_header; /* ICP2.0 access control request header */
    char *w_req_body; /* ICP2.0 access control request body */

    /* Access Control */
    /* host */
    w_host = (char *)ap_table_get( r->headers_in, "HOST" );
    /* url */
    w_content_url = ap_psprintf( r->pool,
                                "%s://%s%s",
                                ap_http_method( r ),
                                w_host,
                                path );

    /* SessionID */
    w_session = cookie + strlen( COOKIENAME_EQ );

    /* Request Message */
    w_req_body = ap_psprintf( r->pool,
                             "MSG_ID: ReqAccessUID\r\nIW_INFO: %s\r\nACC_URL: %s",

```

```

        w_session,
        w_content_url );
ChangeEOF( w_req_body, strlen( w_req_body ) );
w_req_header = ap_psprintf( r->pool,
        "REQ / ICP/2.0\r\nContent-length: %d\r\n\r\n",
        strlen( w_req_body ) );
w_request = ap_pstrcat( r->pool, w_req_header, w_req_body, NULL );

return( w_request );
}

/*-----*/
/*  ERR_NO acquisition from access control response */
/*  */
/*  1. Obtains the ERR_NO value from the body of the response (spaces are
ignored). */
/*  2. Converts the obtained ERR_NO value into a numeric value and returns it. */
/*  */
/*-----*/
int GetErrNo( r, res_body )
request_rec *r; /* Apache request structure pointer */
char *res_body; /* Decrypted ICP2.0 response body section */
{
    char *w_err_no; /* ICP2.0 response err_no (response code) */
    char *w_ptr; /* Pointer to work area */
    char *w_ptr_chr; /* Area for saving characters during work */

    /* err_no */
    w_err_no = ap_pstrdup( r->pool, "" );
    w_ptr = ap_pstrdup( r->pool, strstr( res_body, "ERR_NO:" ) + 7 );
    while( *w_ptr != '\0' && *w_ptr != '\r' ){
        if( *w_ptr == ' ' ){
            w_ptr++;
            continue;
        }
        w_ptr_chr = ap_psprintf( r->pool, "%c", *w_ptr );
        w_err_no = ap_pstrcat( r->pool, w_err_no, w_ptr_chr, NULL );
        w_ptr++;
    }
    return( atoi( w_err_no ) );
}

/*-----*/
/*  HTTP header setting */
/*  */
/*  1. Obtains the user ID. */
/*  2. Obtains the session information. */
/*  3. Sets the HTTP_UID header. */
/*  4. Sets the HTTP_SESSION header. */
/*  */
/*-----*/
void SetHttpHeader( r, res_body, cookie )

```

```

request_rec *r; /* Apache request structure pointer */
char *res_body; /* Decrypted ICP2.0 response body section */
char *cookie; /* Cookie for IceWall */
{
    char *w_uid; /* HTTP_UID setting value string */
    char *w_ptr; /* Pointer to work area */
    char *w_ptr_chr; /* Area for saving characters during work */
    char *w_session; /* HTTP_SESSION setting value string */

    /* UID */
    w_uid = ap_pstrdup( r->pool, "" );
    w_ptr = ap_pstrdup( r->pool, strstr( res_body, "IW_UID:" ) + 7 );
    while( *w_ptr != '\0' && *w_ptr != '\r' ){
        if( *w_ptr == ' ' ){
            w_ptr++;
            continue;
        }
        w_ptr_chr = ap_psprintf( r->pool, "%c", *w_ptr );
        w_uid = ap_pstrcat( r->pool, w_uid, w_ptr_chr, NULL );
        w_ptr++;
    }
    /* SessionID */
    w_session = cookie + strlen( COOKIENAME_EQ );

    /* Set Header */
    ap_table_set( r->headers_in, "Uid", w_uid );
    ap_table_set( r->headers_in, "SESSION", w_session );

    return;
}

/*-----*/
/*      Certd socket communication      */
/*                                     */
/*      1. Creates a socket.           */
/*      2. Connects to certd.         */
/*      3. Sends a request message.    */
/*      4. Obtains a response message. */
/*      5. Closes a socket.           */
/*                                     */
/*-----*/
void SocketCertd( sendsize, request, response )
int sendsize; /* Transfer size */
char *request; /* ICP2.0 request message */
char *response; /* ICP2.0 response message */
{
    int sock; /* Socket */
    struct sockaddr_in addr; /* Socket information */
    struct hostent *hent; /* Host information */

    /* Certd ( localhost:14142 ) */

```



```

memset( &addr, 0x00, sizeof(struct sockaddr_in) );
addr.sin_family = AF_INET;
addr.sin_port = htons( PORT );
hent = gethostbyname( CERT );
memcpy( &addr.sin_addr, hent->h_addr, hent->h_length );

sock = socket( AF_INET, SOCK_STREAM, 0 );

connect( sock, (struct sockaddr*)&addr, sizeof( addr ) );

send( sock, request, sendsize, 0 );

recv( sock, response, 4096, 0 );

shutdown( sock, SHUT_RDWR );
close( sock );
}

/*-----*/
/*  Encryption/decryption */
/*                               */
/*  1. EOF encryption/decryption */
/*                               */
/*-----*/
void ChangeEOF( buff, len )
char *buff; /* Encryption (decryption) target string */
int len; /* String length */
{
    char ascii; /* Lower 4 ASCII bits */
    int count; /* Counter */
    int index; /* Index */

    index = 0;
    for( count = 0; count < len; count++){

        /* ASCII 4bit */
        ascii = CRYPTOKEY[index++] & 0x0F;
        if( index >= 54 ) index = 0;

        /* EOR */
        buff[count] ^= ascii;

        /* 0x7f */
        if( buff[count] == 0x7F ) buff[count] ^= ascii;
    }
}

/* create per-dir config structures */
/*-----*/
/*  httpd.conf setting value structure initialization */
/*                               */
/*  1. Reserves the structure area. */

```

```
/* 2. Sets the initial value. */
/* */
/*-----*/
static void *create_per_dir_config( p, arg )
pool *p; /* Pool */
char *arg; /* Argument */
{
    sample_dir_config *conf; /* httpd.conf setting value structure */

    conf=ap_pccalloc( p,sizeof( sample_dir_config ) );
    conf->dfw_path = ap_pstrdup( p, "" );
    return( (void *)conf );
}

/*-----*/
/* httpd.conf setting value structure acquisition */
/* */
/* 1. Sets the value in DFW_PATH from httpd.conf to conf->dfw_path. */
/* */
/*-----*/
static const char *set_dfw_path( cmd , dummy , arg )
cmd_parms *cmd; /* Command parametr */
void *dummy; /* Dummy void pointer */
char *arg; /* Argument */
{
    sample_dir_config *conf; /* httpd.conf setting value structure */

    conf = dummy;
    conf->dfw_path = ap_pstrdup( cmd->pool, arg );

    return( NULL );
}

/* table of config file commands */
static const command_rec cmds[]=
{
    { "DFW_PATH", set_dfw_path, NULL, OR_ALL, TAKE1, "DFW_PATH" },
    { NULL }
};

/* Dispatch list for API hooks */
module MODULE_VAR_EXPORT sample_module = {
    STANDARD_MODULE_STUFF,
    NULL, /* module initializer */
    create_per_dir_config, /* create per-dir config structures */
    NULL, /* merge per-dir config structures */
    NULL, /* create per-server config structures */
    NULL, /* merge per-server config structures */
    cmds, /* table of config file commands */
    NULL, /* [#8] MIME-typed-dispatched handlers */
    NULL, /* [#1] URI to filename translation */
}
```

```
NULL,          /* [#4] validate user id from request */
NULL,          /* [#5] check if the user is ok _here_ */
SAMPLE_DFW_Main, /* [#3] check access by host address */
NULL,          /* [#6] determine MIME type */
NULL,          /* [#7] pre-run fixups */
NULL,          /* [#9] log a transaction */
NULL,          /* [#2] header parser */
NULL,          /* child_init */
NULL,          /* child_exit */
NULL           /* [#0] post read-request */
#ifdef EAPI
, NULL,        /* EAPI: add_module */
NULL,         /* EAPI: remove_module */
NULL,         /* EAPI: rewrite_command */
NULL          /* EAPI: new_connection */
#endif
};
```

10 Sample Communication Message Encryption Library

The following shows the contents of the communication message encryption library development kit installed by default when you install the Forwarder and Authentication Module.

The development kit for the Forwarder consists of the following directories and files.

Directory		Description
/opt/icewall-ss0/developkit/dfw/DfwCryptoExit	Makefile	Makefile
	iwcrypto.c	Sample source file
	iwcrypto.h	Header file
	libcpt.a	ARI archive file

The development kit for the Authentication Module consists of the following directories and files.

Directory		Description
/opt/icewall-ss0/developkit/certd/CryptoExit	Makefile	Makefile
	iwcrypto.c	Sample source file
	iwcrypto.h	Header file
	libcpt.a	ARI archive file

10.1 Sample source file (iwcrypto.c)

The sample source file below is common to the Forwarder and Authentication Module.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "iwcrypto.h"

#define CRYPTOKEY "ygHxXW7Y+LiQDz7YUyIh4I9Ig28X6VaYAHMmFIXGEnJ
XEAXx7hGQgY"

void EXCPT_EOF( char *buff, int bufflen );

IW_EXCRYPTO *IW_ExEncrypto( char *buff, int bufflen )
{
    IW_EXCRYPTO *excrypto;

    excrypto = IW_ExCPTCreateCrypto( buff, bufflen, 0, NULL );
    EXCPT_EOF( excrypto->buff, excrypto->bufflen );
}
```

```
    return( excrypto );
}

IW_EXCRYPTO *IW_ExDecrypto( char *buff, int buflen )
{
    IW_EXCRYPTO *excrypto;

    excrypto = IW_ExCPTCreateCrypto( buff, buflen, 0, NULL );

    EXCPT_EOF( excrypto->buff, excrypto->buflen );

    return( excrypto );
}

void EXCPT_EOF( char *buff, int buflen )
{
    char ascii;
    int count;
    int index;

    index = 0;
    for( count = 0; count < buflen; count++ ){

        /* ASCII 4bit */
        ascii = CRYPTOKEY[index++] & 0x0F;
        if( index >= 54 ) index = 0;

        /* EOR */
        buff[count] ^= ascii;

        /* 0x7f */
        if( buff[count] == 0x7F ) buff[count] ^= ascii;
    }
}
```

* The sample encryption library may insert NUL (0x00) in an encrypted string. For such encryption, replace the string copy function with some function not to search NUL.

10.2 Header file (iwcrypto.h)

The header file below is common to the Forwarder and Authentication Module.

```
#ifndef IWCRYPTO_H
#define IWCRYPTO_H

typedef struct _IW_EXCRYPTO {
    char *buff;
    int buflen;
    int result;
}
```

```
    char *errmsg;
} IW_EXCRYPTO;

IW_EXCRYPTO *IW_ExEncrypto( char *buff, int buflen );
IW_EXCRYPTO *IW_ExDecrypto( char *buff, int buflen );

IW_EXCRYPTO *IW_ExCPTCreateCrypto( char *buff, int buflen,
                                   int result, char *errmsg );
int IW_ExCPTReleaseCrypto( IW_EXCRYPTO *excrypto );

#endif /* #ifndef IWCRYPTO_H */
```

10.3 Makefile for the Forwarder (HP-UX edition: Makefile) 10.0

```
CC                = cc

CCFLAGS           = -D_UNIX -D_HPUX_SOURCE -D_POSIX_C_SOURCE=19
9506L -Aa +e -D_FILE_OFFSET_BITS=64 -z +DD64 +z

LIBS              = ./libcpt.a

INCLUDE           = -I./

MAKEFILE          = Makefile

OBJS              = iwcrypto.o

PROGRAM           = DfwCryptoExit

SRCS              = iwcrypto.c

all:              $(PROGRAM)

$(PROGRAM):       $(OBJS)
                  ld -b -z -o lib$(PROGRAM).sl $(OBJS) $(LIBS)

.c.o:             $(CC) $(CCFLAGS) $(INCLUDE) -c $(SRCS)

clean:            rm -f $(OBJS) lib$(PROGRAM).sl core
```

10.4 Makefile for the Forwarder (Linux edition: Makefile) 10.0

```
CC                = gcc

CCFLAGS           = -m64 -fPIC -DLinux -D_FILE_OFFSET_BITS=64 -D_LAR
GEFILE_SOURCE -D_GNU_SOURCE

LDLFLAGS          = -m64 -fPIC

LIBS              = ./libcpt.a

INCLUDE           = -I./

MAKEFILE          = Makefile

OBJS              = iwcrypto.o

PROGRAM           = DfwCryptoExit

SRCS              = iwcrypto.c

all:              $(PROGRAM)

$(PROGRAM):       $(OBJS)
gcc -shared $(LDLFLAGS) -o lib$(PROGRAM).sl $(OBJS)

$(LIBS)
.c.o:
$(CC) $(CCFLAGS) $(INCLUDE) -c $(SRCS)

clean:
rm -f $(OBJS) lib$(PROGRAM).sl core
```

10.5 Makefile for the Authentication Module (HP-UX edition: Makefile)

```
CC                = cc

CCFLAGS           = -D_UNIX -D_HPUX_SOURCE -D_POSIX_C_SOURCE=19
9506L -Aa +e -D_FILE_OFFSET_BITS=64 -z +DD64 +z

LIBS              = ./libcpt.a

INCLUDE           = -I./

MAKEFILE          = Makefile

OBJS              = iwcrypto.o
```

PROGRAM	= CryptoExit
SRCS	= iwcrypto.c
all:	\$(PROGRAM)
\$(PROGRAM):	\$(OBJS) ld -b -z -o lib\$(PROGRAM).sl \$(OBJS) \$(LIBS)
.c.o:	\$(CC) \$(CCFLAGS) \$(INCLUDE) -c \$(SRCS)
clean:	rm -f \$(OBJS) lib\$(PROGRAM).sl core

10.6 Makefile for the Authentication Module (Linux edition: Makefile)

CC	= gcc
CCFLAGS	= -m64 -fPIC -DLinux -D_FILE_OFFSET_BITS=64 -D_LAR
GEFILE_SOURCE	-D_GNU_SOURCE
LDFLAGS	= -m64 -fPIC
LIBS	= ./libcpt.a
INCLUDE	= -I./
MAKEFILE	= Makefile
OBJS	= iwcrypto.o
PROGRAM	= CryptoExit
SRCS	= iwcrypto.c
all:	\$(PROGRAM)
\$(PROGRAM):	\$(OBJS) gcc -shared \$(LDFLAGS) -o lib\$(PROGRAM).sl \$(OBJS)
\$(LIBS)	
.c.o:	\$(CC) \$(CCFLAGS) \$(INCLUDE) -c \$(SRCS)
clean:	rm -f \$(OBJS) lib\$(PROGRAM).sl core