FSM Encryption Whitepaper

FileNet System Monitor 4.0.0

FileNet Corporation

Table of Contents

Chapter 1. Notices

This document contains information proprietary to FileNet Corporation (FileNet). Due to continuing product development, product specifications and capabilities are subject to change without notice. You may not disclose or use any proprietary information or reproduce or transmit any part of this document in any form or by any means, electronic or mechanical, for any purpose, without written permission from FileNet.

FileNet has made every effort to keep the information in this document current and accurate as of the date of publication or revision. However, FileNet does not guarantee or imply that this document is error free or accurate with regard to any particular specification. In no event will FileNet be liable for direct, indirect, special incidental, or consequential damages resulting from any defect in the documentation, even if advised of the possibility of such damages. No FileNet agent, dealer, or employee is authorized to make any modification, extension, or addition to the above statements. FileNet may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Furnishing this document does not provide any license to these patents, trademarks, copyrights, or other intellectual property.

Please take a few moments to read the *End User License Agreement* on the FileNet System Monitor 4.0.0 documentation CD. By installing the FileNet System Monitor 4.0.0 software, the customer agrees to be bound by the terms of this agreement. FileNet System Monitor, copyright-protected by CENIT AG Systemhaus, is licensed and rebranded by FileNet Corporation. FileNet, ValueNet, Visual WorkFlo, and OSAR are registered trademarks of FileNet Corporation. Document Warehouse and UserNet are trademarks of FileNet Corporation. All other product and brand names are trademarks or registered trademarks of their respective companies. See the *Centera License Agreement* for copyright information pertaining to EMC Centera.

FileNet Corporation 3565 Harbor Boulevard Costa Mesa, California 92626 USA
800.FILENET (345.3638) Outside the U.S., call: 1.714.327.3400
www.filenet.com (http://www.filenet.com)

Chapter 2. Copyright Notice

FileNet System Monitor

(June, 2007)

Trademarks

Notice

CENIT AG Systemhaus may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

CENIT AG Systemhaus, Product Marketing Tivoli Plus Modules, Industriestr. 52-54, 70565 Stuttgart, Germany

Chapter 3. Preface

This documents intents to explain how the events transmitted between CALA processes can be protected against being attacked by unauthorized persons.

It shows which mechanisms can be used to encrypt the communication between the processes, mentions the advantages and disadvantages of the different mechanisms and shows how they are set up.

Related Documents

FSM CALA Installation and User's Guide

Chapter 4. CALA Encryption

To protect CALA processes from being abused by crackers, there are additional encryption features, which should be used on unsecure connections.

The TCP/IP protocol is used for both local and remote communication between CALA processes. For local communication the loopback device is used, so there is no traffic on the network.

For remote connection, there are 4 security levels available:

- 0 no encryption (is used by default for internal connections over loopback device only)
- 1 vigenere encryption (default, used in CALA 1.0)
- 2 RSA encryption (safe but slow)
- 3 one-time-pad encryption

Encryption Keys

If no special encryption key is given, CALA uses a default key for encrypting events.

It is possible to use the same key for several communication lines or to set up an own encryption key for each line.

The customer can create it s own encryption keys the using the tool `crypttool` which is part of the CALA distribution. It can generate different keys for clients and servers, the key length is freely definable.

The usage of the crypttool is explained later (see the chapter *The crypttool*)

The vigenere algorithm

The vigenere algorithm is a very fast but simple and unsafe encryption algorithm. Each character is encoded with one character from the key. Encoding is done by adding the two charater codes and modulo the result to 255. The encryption key must have the same length as the message to encrypt. If the key is not long enough to encode the message, it is simply repeated.

Example: encrypting the message MESSAGE with the key

Text	Character codes
MESSAGE	77 69 83 83 65 71 69
KEYKEYK	75 69 89 75 69 89 75
ÿè¼Elzpesåá É	152 138 172 158 134 160 144

To decrypt the message, the process is run in reverse.

Note: The vigenere cipher can easily broken by using statistical methods.

Further information about vigenere encryption can be found on the internet:

http://raphael.math.uic.edu/~jeremy/crypt/vignere.html

http://lor.trincoll.edu/~cpsc/cryptography/vigenere.html

RSA encryption

RSA is a safe, but slow encryption algorithm. One of the main advantages of RSA is its asymmetry. There are different keys for en- and decryption.

For CALA this means that the clients only know how to encrypt events and only the server knows how to decrypt them.

So if a hacker gets access to a client system he wouldn t be able to decrypt events from other client systems even if they used the same encryption key as the cracked system.

For information how RSA works see the official RSA homepage at http://www.rsa.com.

The one-time-pad encryption algorithm

The one-time-pad encryption (encryption level 3) uses a combination of the vigenere encryption and the RSA encryption algorithm.

The idea is to combine the advantages of the two methods: the high performance of the vigenere algorithm and the security of RSA.

Combining vigenere and RSA algorithms

The CALA encryption algorithm uses vigenere encryption for event data but every byte is encrypted with a key of its own. The keys are generated at runtime with a special algorithm, which needs only little initial data to be transmitted from client to server. This initial data is sent RSA encrypted.

Because the generated keys iterate after a while, new initial data has to be created if a number of keys has been used.

Generating temporary keys

The temporary keys are generated using a shift register sequence algorithm. Some start values x_1, x_n and some coefficients $a_1, ,a_n$ are generated using the machine dependent random function.

This initial data is transmitted to the server. x_1, x_n are the keys for the first *n* bytes encrypted.

The following keys are calculated using the following algorithm:

$x_n = a_1 * x_1 + \ldots + a_n * x_n$

$x_{n-1} = x_n$

$x_{n-2} = x_{n-1}$

. . .

$x_1 = x_2$

This shift register algorithm is a standard algorithm for generating random data. The generated numbers iterate after at least 2^{n-1} numbers. To ensure safe communication, the keys should not repeat, so the number of keys generated by this algorithm should be less than 2^{n-1}.

The CALA encryption mechanism implements each *x* and *a* as one byte. *N* (the number of start values) can be set as the CALA argument *YS<n>*, the number of keys (bytes) to be created from this start data is specified with the argument *YB<no>*.

The customer can freely decide how often new keys will be generated. A fewer number of keys means that key generation occurs more often.

Implementation

At connection establishment, the client generates (ramdom) initializing data and sends it RSA encrypted[1] to the server (1). With this initial data, both processes calculate a large number of temporary encryption keys (2).

The following communication packages are transmitted vigenere encrypted (4), each package using a number of the temporary keys (depending on its size). After a key is used, it will be discarded. If no more keys are available, a new initialization package is sent to the server (1a).

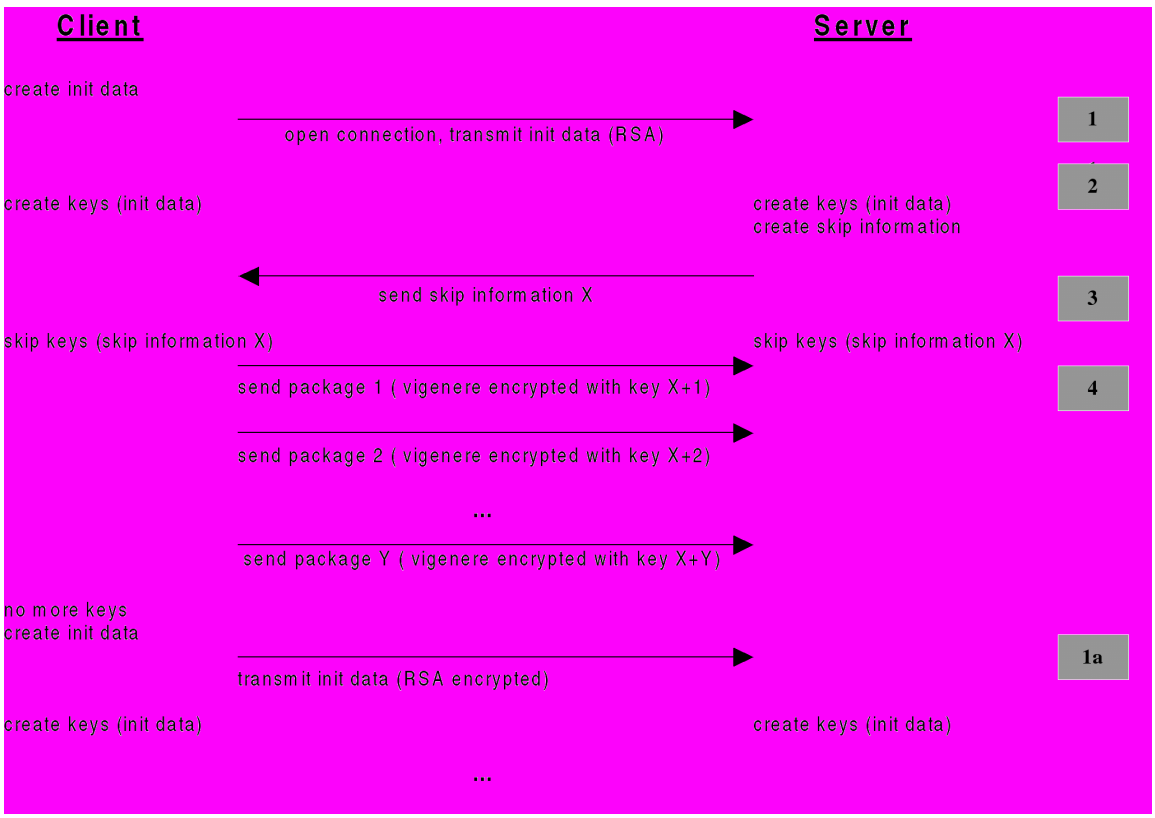

Figure 4-1. Encrypted Client-Server communication.

To ensure that a cracker cannot attack the system by replaying old TCP/IP packages, the server sends a random skip information after creating the keys (3). This tells the client not to start with the first generated key, but with a later one.

1. In fact they are not only RSA encrypted, but RSA and vigenere encrypted with the encryption keys from the key file.

When replaying old TCP/IP packages, this will cause the server to be unable to decrypt the received packages, because the client uses the wrong (old) skip information.

Configuring encryption

Encryption configuration is based on command line arguments for each CALA components.

These parameters can be specified using CALA-GUI by adding values to the text field supplementary parameters

The following parameters are supported:

parameter	example	Description
-YK<keyfile>	*-YKcala_key*	This parameter specifies the name of the key file containing the encryption keys to use. If the keyfile name contains a *%s*, this is replaced by the hostname or ip-address of the machine connected to. In this way you can have different key files for different connections. By default the key is searched in `.cala_key` within the CALA directory. If no keyfile is found, a standard key is used. This parameter has to be set on the client side as well as on the server side.
-YL<level>	*-YL1*	This is a parameter to be set on the client side. It specifies which encryption level is to be used for outgoing connections (see list of possible connection levels above). By default a encryption level of *1* is used for remote connections. (Local connections always use encryption level 0). If using an unsecure connection, an encryption level of 3 is recommended.
-YA<level>	*-YA1*	This server-side parameter controls the minimum encryption level a client has to use when connecting to the server. Clients using a lower connection level are refused. The minimum encryption level for accepting connections is *1* by default. For servers with unsecure connection, a level of 3 is recommended.
-YE	*-YE*	*CALA Version 1.2* This parameter *disables* the Encryption Error Event creation. When a component receives a packet that could not be decrypted (for example if the cryptkeys do not match) an event is created and sent to all targets indicating that this kind of error occurred and who sent this package.

On critical connections which may be attacked by hackers the usage of level 3 is recommended.

For encryption level 3 (one time pad encryption) there are two additional parameters. These parameters must be given on the client s command line. The servers can handle different start values and keys from different clients.

parameter	Example	Description
-YS<no>	*-YS200*	This parameter specifies the number of start values to be transmitted to the server for the key generation algorithm. A value between 100 and 200 is pretty safe. The default value is *YS100*.
-YB<no>	*-YB1000000*	This parameter specifies the number of keys (bytes) to be created from the start values given with YS parameter. To have no iteration within the generated keys, a value less than 2_{n-1} (n=number given with YS) is recommended. The default value is *YB1000000*.

The crypttool

Encryption keys can be generated using the tool crypttool

This is the usage screen (call **crypttool** *-?* to get this):

```
****************************************************************
**                                                            **
** crpt_tool is part of the CENIT Advanced Logfile Adapter    **
**                                                            **
**                                                            **
** version: 2.02-034 - generation date: Oct 13 2005 17:04:24  **
**                                                            **
**                                                            **
** (c) 1999-2005 CENIT AG Systemhaus                          **
**                                                            **
****************************************************************
unknown paramter: "-?"
usage: ./crypttool.exe [-c <filename>] [-s <filename>]
[-a <filename>] [-v <length>] [-r <length>]
-c, --client-key: filename of client key-file to write
-s, --server-key: filename of server key-file to write
-a, --all: filename of complete key-file to write - default: '.cala_key'
-v, --vigenerekey-len: length of vigenerkey (byte) - default: 2000
-r, --rsakey-len: length of rsa-key (decimal places) - default: 154, maximum: 154
```

By default, the program generates a file named `.cala_key` containing three keys:

- a vigenere key for encryption and decryption (the same key is used for both operations), 2000 bytes long
- a RSA key for encryption, using 154 decimal places
- a RSA key for decryption, using 154 decimal places

To get higher security, the RSA keys for encryption and decryption should be splitted. The client processes only need to encrypt the data, the servers only need to decrypt it.

> **Note:** The crypttool uses system specific random functions to create the keys, so if called two times with the same parameters, different keys are generated. If using separate keys for clients and servers, ensure these keys are generated during the same program run.

The crypttool supports the following arguments:

parameter	Example	Description
-c <keyfile>	*-c client_key*	A client key (encryption only) is created and saved into the given file.
-s <keyfile>	*-s server_key*	A server key (decryption only) is created and saved into the given file.
-a <keyfile>	*-a complete_key*	A complete key (encryption and decryption) is created and saved into the given file.
-v <bytes>	*-v 8000*	Sets the length (in byte) of the vigenere key to create.
-r <length>	*-r 100*	Sets the length (in decimal places) of the prime numbers used by the RSA key to create. (Maximum: 154)

> **Important:** The maximum length for RSA keys is 154 decimal places. If a higher value is given, the program creates a key with the maximum length of 154. (For comparison: a 512 bit number has up to 155 decimal places.)