Secure Provisioning Data Transport using Http

Confidential, Draft
RFC 36

13 Pages

Abstract

We describe a remote method for performing Secure Provisioning Data Transport to a Service Platform. The method is using the http and https protocols for communication between the Service Platform and the operator's server(s). The methods may optionally make use of a shared secret to ensure that necessary parts are secure, even if the communication between the Service Platform and the operator is done over public networks.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies Secure Provisioning Data Transport using Http, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

Frank Seliger (IBM) and Ventzi Nikov (Acunia) have made significant contributions to the contents of this document.

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
1.00A	September 17, 2001	Initial revision. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00B	September 18, 2001	SSL client authentication added and corresponding certificate transfer also included. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00C	September 20, 2001	Allow both https and http with data encryption for the Load phase. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00D	September 20, 2001	Certificate transfer removed from Setup phase. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00E	September 21, 2001	Terminology aligned with new version of RFC 27. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00F	October 21, 2001	Shared secret definitions moved from RFC 27. Aligned with changes of RFC 27. Protocol, using shared secret, modified and a lot more details are included. Author: Lars-Erik Helander, Gatespace <helander@gatespace.com>.
1.00G	October 29, 2001	Updates after comments. Some re-structuring of the document. Content coding details. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.
1.00H	October 30, 2001	Updates after comments. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.

1.00I	November 1, 2001	Correcting MIME details. Certificates made into X.509 variants Certificate details moved to RFC 27. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.
1.00 J	November 30, 2001	1. Dictionary data transport format moved to RFC 27. 2. Server FG re-introduced 3. Authentication MAC is based on encrypted (instead of unencrypted) response data. 4. Minor adjustments. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.
1.00 K	January 4, 2002	1. Aligned with terminology of RFC 20 2. Aligned with terminology of RFC 27 3. X509 issues moved from RFC 27 4. Minor adjustments 5. The DN of SP certificate is specified to equal SP id. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.
1.00 L	August 30, 2002	1. Text adjusted to current state of spec for R3. Authors: Lars-Erik Helander, Gatespace <helander@gatespace.com> and Ben Reed, IBM <breed@almaden.ibm.com>.
1.00 M	September 24, 2002	Text adjusted to current state of spec for R3.

1 Introduction

This document proposes a remote communication method used to perform the Provisioning Data Assignments to a Service Platform. The solution is based on using http or http-based protocols for the Provisioning Data Trasnports. The http-based protocols supported by this RFC are https and a protocol defined in this document, called RSH. Both https and RSH are suitable to use if communication takes place over some public network. The RSH protocol encrypts the payload data and performs necessary authentication, but still use http as transport.

For full context and background information, see [2].

2 Motivation and Rationale

- HTTP – May be used when the data exchange takes place over networks that are secured by other means, such as a VPN or a physically isolated network. Otherwise, HTTP is not a valid scheme because no authentication takes place.
- HTTPS – May be used if the Service Platform is equipped with appropriate certificates.

HTTP and HTTPS share the following qualities:

- Both are well known and widely used
- Numerous implementations of the protocols exist
- Caching of the Management Agent will be desired in many implementations where limited bandwidth is an issue.
- Both HTTP and HTTPS already contain an accepted protocol for caching.
- Both HTTP and HTTPS must be used with the GET method. The response is a ZIP file, implying that the response header Content-Type header must contain application/zip.

3 Technical Discussion

This document describes a number of ways to perform Provisioning Data Transport for the Provisioning Data Assignment protocol defined in [2]. This chapter describes the different transport protocols and how the application protocol are mapped on top of these transport protocols.

3.1 Mapping To HTTP(S) Scheme

This section defines how HTTP and HTTPS URLs must be used with the Initial Provisioning specification.

3.1.1 HTTPS Certificates

In order to use HTTPS, certificates must be in place. These certificates, that are used to establish trust towards the Operator, may be made available to the Service Platform using the Provisioning Service. The root certificate should be assigned to the Provisioning Dictionary before the HTTPS pro-vider is used. Additionally, the Service Platform should be equipped with a Service Platform certificate that allows the Service Platform to properly authenticate itself towards the Operator. This specification does not state how this certificate gets installed into the Service Platform.

The root certificate is stored in the Provisioning Dictionary under the key:

PROVISIONING_ROOTX509 – The Root X.509 Certificate holds certificates used to represent a handle to a common base for establishing trust. The certificates are typically used when authenticating a Remote Manager to the Service Platform. In this case, a Root X.509 certificate must be part of a certificate chain for the Operator's certificate.

3.1.2 Certificate Encoding

Root X.509 certificates are X.509 certificates. Each individual certificate is stored as a byte[] object. This byte[] object is encoded in the default Java manner, as follows:

- The original, binary certificate data is DER encoded
- The DER encoded data is encoded into base64 to make it text.
- The base64 encoded data is prefixed with

  -----BEGIN CERTIFICATE-----
- and suffixed with:

  -----END CERTIFICATE-----

If a record contains more than one certificate, they are simply appended one after the other, each with a delimiting prefix and suffix.

The decoding of such a certificate may be done with the java.security.cert.CertificateFactory class:

```
InputStream bis = new ByteArrayInputStream(x509); // byte[]
CertificateFactory cf = CertificateFactory.getInstance("X.509");
Collection c = cf.generateCertificates(fis);
Iterator i = c.iterator();
while (i.hasNext()) {
  Certificate cert = (Certificate)i.next();
  System.out.println(cert);
}
```

3.1.3 URL Encoding

The URL must contain the Service Platform Identity, and may contain more parameters. These parameters are encoded in the URL according to the HTTP(S) URL scheme. A base URL may be set by an end user but the Provisioning Service must add the Service Platform Identifier, and in case of RSH, a nonce.

If the request URL already contains HTTP parameters (if there is a '?' in the request), the service_platform_id is appended to this URL as an additional parameter. If, on the other hand, the request URL does not contain any HTTP parameters, the service_platform_id will be appended to the URL after a '?', becoming the first HTTP parameter. The following two examples show these two variants:

http://server.operator.com/service-x?foo=bar&service_platform_id=VIN:123456789

http://server.operator.com/service-x?service_platform_id=VIN:123456789

Proper URL encoding must be applied when the URL contains characters that are not allowed.

3.2 Mapping To RSH Scheme

The RSH protocol is an OSGi-specific protocol, and is included in this specification because it is optimized for Initial Provisioning. It requires a shared secret between the management system and the Service Platform that is small enough to be entered by the User.

RSH bases authentication and encryption on Message Authentication Codes (MACs) that have been derived from a secret that is shared between the Service Platform and the Operator prior to the start of the protocol execution.

The protocol is based on an ordinary HTTP GET request/response, in which the request must be signed and the response must be encrypted and authenticated. Both the signature and encryption key are derived from the shared secret using Hashed Message Access Codes (HMAC) functions.

As additional input to the HMAC calculations, one client-generated nonce and one server-generated nonce are used to prevent replay attacks. The nonces are fairly large random numbers that must be generated in relation to each invocation of the protocol, in order to guarantee freshness. These nonces are called clientfg (client-generated freshness guarantee) and serverfg (server-generated freshness guarantee). The major characteristic of the clientfg nonce is uniqueness, and for the serverfg nonce, the main characteristic is randomness. The reason for this distinction is that it is hard to create good random generators, and this way only the servers have to deal with this problem.

In order to separate the HMAC calculations for authentication and encryption, each is based on a different constant value. These constants are called the authentication constant and the encryption constant.

From an abstract perspective, the protocol may be described as follows:

- ψ – Shared secret, 160 bits or more
- s – Server nonce, called servercfg, 128 bits
- c – Client nonce, called clientfg, 128 bits
- Ka – Authentication key, 160 bits
- Ke – Encryption key, 192 bits
- r – Response data
- e – Encrypted data
- E – Encryption constant, a byte[] of 05, 36, 54, 70, 00 (hex)
- A – Authentication constant, a byte[] of 00, 4f, 53, 47, 49 (hex)
- M – Message material, used for Ke calculation.
- m – The calculated message authentication code.
- 3DES – Triple DES, encryption function, see [103] 3DES. The bytes of the key must be set to odd parity.
- IV – Initialization vector for 3DES.
- SHA1 – Secure Hash Algorithm to generate the Hashed Message Authentication Code, see [104] SHA-1. The function takes a single parameter, the block to be worked upon.
- HMAC – The function that calculates a message authentication code, which must HMAC-SHA1. HMAC-SHA1 is defined in [96] HMAC: Keyed-Hashing for Message Authentication. The HMAC function takes a key and a block to be worked upon as arguments.
- {} – Concatenates its arguments
- [] – Indicates access to a sub-part of a variable, in bytes. Index starts at one, not zero.

In each step, the emphasized server or client indicates the context of the calculation. If both are used at the same time, each variable will have server or client as a subcript.

1. The client generates a nonce, stores it somewhere and denotes it clientfg

c = simplenonce

2. The client sends the request with the clientfg to the server.

c (client) -> c(server)

3. The server generates a nonce and denotes it serverfg.

s = randomnonce

4. The server calculates an authentication key based on the SHA1 function, the shared secret, the received clientfg, the serverfg and the authentication constant.

Ka <- SHA1({ψ,c,s,A})

5. The server calculates an encryption key using an SHA-1 function, the shared secret, the received clientfg, the serverfg and the encryption constant. It must first calculate the key material M.

M[1,20] <- SHA1({ψ,c,s,E})

M[21,40] <- SHA1({ψ,M[1,20],c,s,E})

6. The key for DES consists K_e and IV.

Ke <- M[1,24]
IV <- M[25,32]

The server encrypts the response data using the encryption key derived in 5. The encryption algorithm that must be used to encrypt/decrypt the response data is 3DES. 24 bytes (192 bits) from M are used to generate K_e, but the low order bit of each byte must be used as an odd parity bit. This means that before using K_e, each byte must be processed to set the low order bit so that the byte has odd parity.

The encryption/decryption key used is specified by the following: e <- 3DES(Ke,IV,r)

7. The server calculates a MAC m using the HMAC function, the encrypted response data and the authentication key derived in 4.

m <- HMAC(Ka,e)

8. The server sends a response to the client containing the serverfg, the MAC m and the encrypted response data

s(server) => s(client)

m(server) => m(client)

e(server) => e(client)

The client calculates the encryption key K_e the same way the server did in step 5 and 6. and uses this to decrypt the encrypted response data. The serverfg value received in the response is used in the calculation.

r <- 3DES(Ke,IV,e)

9. The client performs the calculation of the MAC m' in the same way the server did, and checks that the results match the received MAC m. If they do not match, further processing is discarded. The serverfg value received in the response is used in the calculation.

Ka <- SHA1({Ψ,c,s,A})

m(prim) <- HMAC(Ka,e)

$m(prim) = m$

3.2.1 Shared Secret

The shared secret should be a key of length 160 bits (20 bytes) or more. The length is selected to match the output of the selected hash algorithm .

In some scenarios, the shared secret is generated by the Operator and communicated to the User, who inserts the secret into the Service Platform through some unspecified means.

The opposite is also possible: the shared secret can be stored within the Service Platform, extracted from it, and then communicated to the Operator. In this scenario, the source of the shared secret could be either the Service Platform or the Operator.

In order for the server to calculate the authentication and encryption keys, it requires the proper shared secret. The server must have access to many different shared secrets, one for each Service Platform it is to support. To be able to resolve this

issue, the server must typically also have access to the Service Platform Identifier of the Service Platform. The normal way for the server to know the Service Platform Identifier is through the application protocol, as this value is part of the URL encoded parameters of the HTTP,HTTPS, or RSH mapping of the Initial Provisioning.

In order to be able to switch Operators, a new shared secret must be used. The new secret may be generated by the new Operator and then inserted into the Service Platform device using a mechanism not covered by this specification. Or the device itself may generate the new secret and convey it

to the owner of the device using a display device or read out, which is then communicated to the new operator out-of-band. Additionally, the generation of the new secret may be triggered by some external event, like holding down the reset button for a specified amount of time.

3.2.2 Request Coding

RSH is mapped to HTTP or HTTPS. Thus, the request parameters are URL encoded. RSH requires an additional parameter in the URL: the clientfg parameter. This parameter is a nonce that is used to counter replay attacks.

3.2.3 Response Coding

The server's response to the client is composed of three parts:

- A header containing the protocol version and the serverfg
- The MAC
- The encrypted response

These three items are packaged into a binary container according to the following table

Bytes	Description	Value hex
4	Number of bytes in header	12
2	Version	01 00
16	serverfg	...
4	Number of bytes in MAC	10
16	Message Authentication Code	MAC
4	Number of bytes of encrypted ZIP file	N
N	Encrypted ZIP file	...

The response content type is an RSH-specific encrypted ZIP file, implying that the response header Content-Type must be application/x-rsh.

3.2.4 RSH URL

The RSH URL must be used internally within the Service Platform to indicate the usage of RSH for initial provisioning. The RSH URL format is identical to the HTTP URL format, except that the scheme is rsh: instead of http:. For example:

rsh://server.operator.com/service-x?foo=bar&service_platform_id=VIN:123456789

3.2.5 Extensions to the Provisioning Service Dictionary

RSH specifies one additional entry for the Provisioning Dictionary:

- PROVISIONING_RSH_SECRET

The value of this entry is a byte[] containing the shared secret used by the RSH protocol.

3.2.6 RSH Transport

RSH is mapped to HTTP or HTTPS and follows the same URL encoding rules, except that the clientfg is additionally coded in the URL:

The clientfg parameter is transported as an HTTP parameter that is appended after the service_platform_id parameter. The second example above would then be:

rsh://server.operator.com/service-x?service_platform_id=VIN:123456789

Which, when mapped to http, becomes

http://server.operator.com/service-x?
service_platform_id=VIN:123456789&clientfg=AHPmWcw%2FsiWYC37xZNdkvQ%3D%3D

4 Security Considerations

Whether message-based or connection-based, the communications used for Initial Provisioning must support mutual authentication and message integrity checking, at a minimum.

By using both server and client authentication in HTTPS, the problem of establishing identity is solved. In addition, HTTPS will encrypt the transmitted data. HTTPS requires a Public Key Infrastructure implementation in order to retrieve the required certificates.

When RSH is used, it is vital that the shared secret is shared only between the Operator and the Service Platform, and no one else.

5 Document Support

5.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt

[2]. Helander, L-E, RFC Management Agent Deployment, OSGi RFC 27, 2001. http://membercvs.osgi.org/rfcs/rfc0027/rfc_0027_InitialProvisioning.pdf

[3]. Krawczyk ,et. al., HMAC: Keyed-Hashing for Message Authentication,IETF RFC 2104, 1997.

[4]. NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.

[5]. Fielding, R., et. al., Hypertext Transfer Protocol – HTTP/1.1, IETF RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[6]. Rescorla, E., HTTP over TLS, IETF RFC 2818, May 2000, http://www.ietf.org/rfc/rfc2818.txt.

5.2 Author's Address

Name	Lars-Erik Helander
Company	Gatespace AB
Address	Stora Badhusgatan 18-20, Gothenburg, Sweden
Voice	+46 31 743 98 43
e-mail	helander@gatespace.com

Name	Ben Reed
Company	IBM

Address	
Voice	
e-mail	breed@almaden.ibm.com

5.3 End of Document