Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 20, 2019

W. Kumari
Google
C. Doyle
Juniper Networks

January 16, 2019

Secure Device Install
draft-wkumari-opsawg-sdi-03

Abstract

Deploying a new network device often requires that an employee physically travel to a datacenter to perform the initial install and configuration, even in shared datacenters with "smart-hands" type support. In many cases, this could be avoided if there were a standard, secure way to initially provision the devices.

This document extends existing auto-install / Zero-Touch Provisioning to make the process more secure.

[Ed note: Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in Github at: https://github.com/wkumari/draft-wkumari-opsawg-sdi. The most recent version of the document, open issues, etc should all be available here. The authors (gratefully) accept pull requests.]

[Ed note: This document introduces concepts and serves as the basic for discussion - because of this, it is conversational, and would need to be firmed up before being published]

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 20, 2019.

Copyright Notice

Table of Contents

1. Introduction

In a growing, global network, significant amounts of time and money are spent simply deploying new devices and "forklift" upgrading existing devices. In many cases, these devices are in shared datacenters (for example, Internet Exchange Points (IXP) or "carrier neutral datacenters"), which have staff on hand that can be contracted to perform tasks including physical installs, device reboots, loading initial configurations, etc. There are also a number of (often vendor proprietary) protocols to perform initial device installs and configurations - for example, many network devices will attempt to use DHCP to get an IP address and configuration server, and then fetch and install a configuration when they are first powered on.

Network device configurations contain a significant amount of security related and / or proprietary information (for example, RADIUS or TACACS secrets). Exposing these to a third party to load onto a new device (or using an auto-install techniques which fetch an (unencrypted) config file via something like TFTP) is simply not acceptable to many operators, and so they have to send employees to remote locations to perform the initial configuration work. As well as having a significant monetary cost, it also takes significantly longer to install devices and is generally inefficient.

There are some workarounds to this, such as asking the vendor to pre-configure the devices before shipping it; asking the smart-hands to

install a terminal server; providing a minimal, unsecured configuration and using that to bootstrap to a complete configuration, etc; but these are often clumsy and have security issues - for example, in the terminal server case, the console port connection could be easily snooped.

This document layers security onto existing auto-install solutions to provide a secure method to initially configure new devices.

1.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Overview / Example Scenario

Sirius Cybernetics Corp needs another peering router, and so they order another router from Acme Network Widgets, to be drop-shipped to a POP. Acme begins assembling the new device, and tells Sirius what the new device's serial number will be (SN:17894321). During the initial boot / testing, the router generates a public-private keypair, and Acme publishes it on their keyserver (in a certificate, for ease of use).

While Acme is shipping the new device, Sirius begins generating the initial device configuration. Once the config is ready, Sirius contacts the Acme keyserver, provides the serial number of the new device and fetches the device's public key. Sirius then encrypts the device configuration and puts this encrypted config on a (local) TFTP server.

When the POP receives the new device, they install it in Sirius' rack, and connect the cables as instructed. The new device powers up and discovers that it has not yet been configured. It enters its autoboot state, and begins DHCPing. Sirius' DHCP server provides it with an IP address and the address of the configuration server. The router uses TFTP to fetch a file named according to its serial number

(acme_17894321.cfg). It then uses its private key to decrypt this file, and, assuming it validates, installs the new configuration.

Only the "correct" device will have the required private key and be able to decrypt and use the config file (See Security Considerations). An attacker would be able to connect to the network and get an IP address. They would also be able to retrieve (encrypted) config files by guessing serial numbers (or perhaps the server would allow directory listing), but without the private keys an attacker will not be able to decrypt the files.

[Ed note: This example uses TFTP because that is what many vendors use in their auto-install / ZTP feature. It could easily instead be HTTP, FTP, etc.]

3. Vendor Role / Requirements

This section describes the vendors roles and responsibilities and provides an overview of what the device needs to do.

3.1. CA Infrastructure

The vendor needs to run some (simple) CA infrastructure to sign and publish certificates. When a device is initially powered on (in the factory) it will generate a public / private keypair and a Certificate Signing Request (CSR), with the commonName being the Serial Number of the device. The device sends this CSR to the CA, which signs the CSR, returns the certificate to the device and also sends it to a certificate publication server.

3.2. Certificate Publication Server

The certificate publication server contains a database of all signed certificates. Customers (e.g Sirius Cybernetics Corp) query this server with a serial number, and retrieve the associated certificate. It is expected that operators will receive the serial numbers of newly purchased devices when they purchase them, and that some

automated system will download and store / cache the certificate. This means that there is not a hard requirement on the uptime / reachability of the certificate publication server.

[Ed: The vendor may not want to expose (for commercial reasons) how many devices it has made. This can be mitigated by using non-contiguous serial numbers, and simply creating "fake devices", etc.]

3.3. Initial Device Boot

When the device is powered on for the very first time, it will generate its keypair. It then generates a CSR (including the device serial number) and sends it to the vendor's CA, which signs the certificate. The device receives the signed certificate and stores it.

3.4. Subsequent Boots

After the initial boot, it the device has no (valid) configuration file, it will perform standard an auto-install type functionality. For example, it will perform DHCP Discovery until it gets a DHCP offer including DHCP option 66 or 150. It will contact the server listed in these DHCP options and download a configuration file named config_<serial_number>.cfg. This is all existing (often vendor proprietary) functionality.

After retrieving the config file, Secure Device Install devices will attempt to decrypt the configuration file using its private key. If it is able to decrypt and validate the file it will install the configuration, and start using it.

[Ed note: SDI will also allows additional functionality, like always storing the configs encrypted, having the device store its config encrypted in flash (so that e.g RMAing a routing engine will not leak config, etc. I'm not describing this in detail because:

1. I want to keep this document simple and focused and, more importantly

2. I left converting this into ID format until the draft cut-off and have run out of time :-)]

4. Operator Role / Responsibilities

4.1. Administrative

When purchasing a new device, the accounting department will need to get the serial number of the new device and communicate it to the operations group.

4.2. Technical

The operator will contact the vendor's publication server, and download the certificate (by providing the serial number of the device). They will then encrypt the initial configuration to that key, and place it on the TFTP server, named config_<SN>.enc. See Appendix B for examples.

5. Future enhancements / Discussion

[Ed note: Ed / RFC Editor to remove this section before publication.]

5.1. Key storage

Currently most network devices will store the private key in NV storage (NVRAM / Flash / Disk), but some vendors are already planning on including a TPM module in their devices. Ideally, the keypair would be stored in a TPM on something which is identified as the "router" - for example, the chassis / backplane. This is so that a keypair is bound to what humans think of as the "device", and not, for example, (redundant) routing engines.

5.2. Key replacement

It is anticipated that some operator may want to replace the (vendor provided) keys after installing the device. This would remove (some) concerns that the vendor may have kept a copy of the private key, or that the device may have been intercepted during shipping and the private key duplicated. This would also allow for the use of certificates signed by the operator's CA (e.g using RFC7030 -

Enrollment over Secure Transport) this is a trivial operation, but is not described here (to avoid cluttering up the doc).

5.3. Device reinstall

Increasingly, operations is moving towards an automated model of device management, whereby portions (or the entire) configuration is programmatically generated. This means that operators may want to generate an entire configuration after the device has been initially installed and ask the device to load and use this new configuration. It is expected (but not defined in this document, as it is too vendor specific) that vendors will allow the operator to e.g scp a new, encrypted config (or part of a config) onto a device and then request that the device decrypt and install it (e.g: 'load replace <filename> encrypted)).

6. IANA Considerations

This document contains no IANA considerations.Template: Fill this in!

7. Security Considerations

This needs to be completed, including:

1. We are trusting the vendor to have not kept a copy of the private key when the device initially generated its keypair. Unfortunately you are already trusting the vendor in many ways - it could have included a backdoor in it's code, etc.

2. Devices should be storing their keying information in something like a TPM, to help mitigate the private key being extracted (e.g read off disk) in shipping, when the device is first unpacked by smart-hands, etc). A number of vendors already include a TPM for other security functions.

8. Acknowledgements

The authors wish to thank some folk, including Benoit Claise, Colin Doyle, Sam Ribeiro, and Sean Turner.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[I-D.ietf-sidr-iana-objects] Manderson, T., Vegoda, L., and S. Kent, "RPKI Objects issued by IANA", draft-ietf-sidr-iana-objects-03 (work in progress), May 2011.

Appendix A. Changes / Author Notes.

[RFC Editor: Please remove this section before publication]

From -00 to -01

- Nothing changed in the template!

Appendix B. Demo / proof of concept

This section contains a rough demo / proof of concept of the system. It is only intended for illustration; presumably things like algorithms, key lengths, format / containers will provide much fodder for discussion.

It uses OpenSSL from the command line, in production something more automated would be used. In this example, the serial number of the router is SN19842256.

B.1. Step 1: Generating the certificate.

This step is performed by the router. It generates a key, then a csr, and then a self signed certificate.

B.1.1. Step 1.1: Generate the private key.

```
$ openssl genrsa -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................
.....................................................
.........................+++
...................+++
e is 65537 (0x10001)
```

B.1.2. Step 1.2: Generate the certificate signing request.

```
$ openssl req -new -key key.pem -out SN19842256.csr
Country Name (2 letter code) [AU]:.
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:SN19842256
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:.
```

B.1.3. Step 1.3: Generate the (self signed) certificate itself.

```
$ openssl req -x509 -days 36500 -key key.pem -in SN19842256.csr -out
SN19842256.crt
```

The router then sends the key to the vendor's keyserver for publication (not shown).

B.2. Step 2: Generating the encrypted config.

The operator now wants to deploy the new router.

They generate the initial config (using whatever magic tool generates router configs!), fetch the router's certificate and encrypt the config file to that key. This is done by the operator.

B.2.1. Step 2.1: Fetch the certificate.

```
$ wget http://keyserv.example.net/certificates/SN19842256.crt
```

B.2.2. Step 2.2: Encrypt the config file.

I'm using S/MIME because it is simple to demonstrate. This is almost definitely not the best way to do this.

```
$ openssl smime -encrypt -aes-256-cbc -in SN19842256.cfg\
  -out SN19842256.enc -outform PEM SN19842256.crt
$ more SN19842256.enc
-----BEGIN PKCS7-----
MIICigYJKoZIhvcNAQcDoIICezCCAncCAQAxggE+MIIBOgIBADAiMBUxEzARBgNV
BAMMClNOMTk4NDIyNTYCCQDJVuBlaTOb1DANBgkqhkiG9w0BAQEFAASCAQBABvM3
...
LZoq08jqlWhZZWhTKs4XPGHUdmnZRYIP8KXyEtHt
-----END PKCS7-----
```

B.2.3. Step 2.3: Copy config to the config server.

```
$ scp SN19842256.enc config.example.com:/tftpboot
```

B.3. Step 3: Decrypting and using the config.

When the router connects to the operator's network it will detect that does not have a valid configuration file, and will start the "autoboot" process. This is a well documented process, but the high level overview is that it will use DHCP to obtain an IP address and config server. It will then use TFTP to download a configuration file, based upon its serial number (this document modifies the solution to fetch an encrypted config file (ending in .enc)). It will then then decrypt the config file, and install it.

B.3.1. Step 3.1: Fetch encrypted config file from config server.

```
$ tftp 192.0.2.1 -c get SN19842256.enc
```

B.3.2. Step 3.2: Decrypt and use the config.

```
$ openssl smime -decrypt -in SN19842256.enc -inform pkcs7\
  -out config.cfg -inkey key.pem
```

If an attacker does not have the correct key, they will not be able to decrypt the config:


```
$ openssl smime -decrypt -in SN19842256.enc -inform pkcs7\
  -out config.cfg -inkey wrongkey.pem
Error decrypting PKCS#7 structure
140352450692760:error:06065064:digital envelope
 routines:EVP_DecryptFinal_ex:bad decrypt:evp_enc.c:592:
$ echo $?
4
```

Authors' Addresses

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US

Email: warren@kumari.net

Colin Doyle
Juniper Networks
1133 Innovation Way
Sunnyvale, CA 94089
US

Email: cdoyle@juniper.net