14G Support for HTTP and HTTPS across iDRAC9 with Lifecycle Controller Interfaces

Harness HTTP/HTTPS to inventory, provision, monitor, and update the 14th generation PowerEdge servers.

Dell Engineering
June 2017

Authors

Paul Rubin, Sr. Product Manager
John Paul Harvey, Firmware Senior Principal Engineer
Rohitkumar Arehalli, Firmware Engineer

A Dell EMC Technical White Paper

Revisions

Date	Description
June 2017	Initial release

Contents

Executive summary

This technical white paper describes the enhanced support for HTTP and HTTPS file services in the 14th generation Dell EMC PowerEdge servers. These enhancements enable iDRAC9 with Lifecycle Controller embedded management automation for server inventory, provisioning, monitoring, and update to operate by using performant and secure HTTP and HTTPS-based file sharing.

It describes the facilities within iDRAC9 with Lifecycle Controller and also provides information about security considerations, network, server, proxy, and configuration along with key resources and information.

1 Introduction

Enabling web technology-based data center automation, 14th generation Dell EMC PowerEdge servers provide enhanced embedded management automation by using HTTP and HTTPS.

14th generation PowerEdge HTTP and HTTPS support, including proxy support, has been extended to BIOS UEFI HTTP Boot, iDRAC9 Auto Config, the Lifecycle Controller User Interface (UI), the iDRAC Graphical User Interface (GUI), RACADM CLI, WS-Man, and Redfish APIs, thus enhancing the automation of server lifecycle management.

1.1 Overview of 14th generation PowerEdge HTTP/HTTPs support

Some of these additions are covered in other whitepapers. For the most detailed information see the feature documentation.

1.1.1 BIOS UEFI HTTP Boot

14th generation of PowerEdge server BIOS supports Unified Extensible Firmware Interface (UEFI) HTTP Boot. HTTP boot is client-server communication-based application which uses DHCP, DNS, and HTTP protocols to provide the capability for system deployment and configuration over the network. HTTP boot is a replacement of PXE boot, with higher security and more reliable performance.

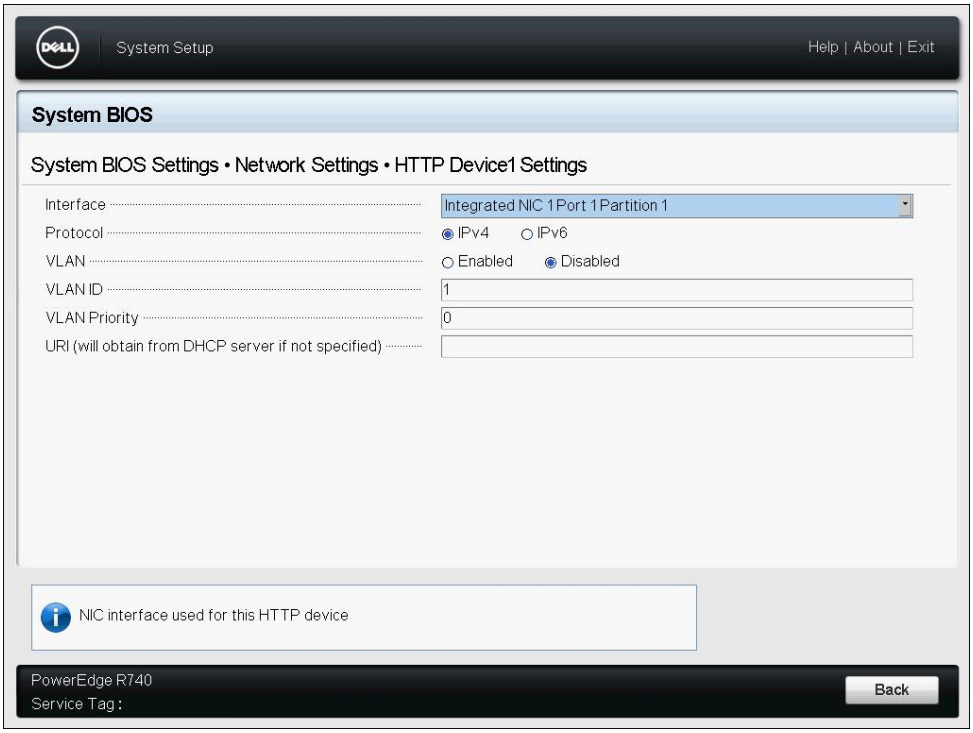

Figure 1 14G BIOS configuration for UEFI HTTP Boot

Similarly to PXE, 14G BIOS enables the bootstrap of operating system (OS) images stored on a network file share. HTTP Boot is configured by using the BIOS attributes that describe the network interface and settings required to access a specified boot image stored on an HTTP server. HTTP Boot attributes are configurable by using the BIOS UI, the iDRAC9 GUI, RACADM CLI, WS-Man, and Redfish APIs. For more information about HTTP Boot and 14G BIOS, see the relevant *Owner's Manual* of the 14th generation of PowerEdge servers available on the support site.

1.1.2 iDRAC9 Auto Config

iDRAC firmware for 12th, 13th, and 14th generation PowerEdge servers provide Auto Config, a complete "zero-touch" mechanism for configuring a bare-metal server from a common server configuration profile. This feature allows IT admins to build an environment in which servers can automatically configure all hardware settings using existing data center infrastructure. This removes the need for high-touch, manual steps to configure server subsystems such as storage, networking, and BIOS. Administrators can develop configuration profiles for classes of servers and apply those profiles without interacting with individual systems.

For 14th generation of PowerEdge servers, iDRAC9 Auto Config has been updated to enable delivery of server configuration profiles via HTTP and HTTPS. For more information see the whitepaper *Zero-Touch Bare Metal Server Provisioning using Dell iDRAC with Lifecycle Controller Auto Config*.

1.1.3 Lifecycle Controller UI

iDRAC9 with Lifecycle Controller firmware version 3.00.00.00 enables HTTP support for a range of features for 14th generation PowerEdge servers. iDRAC9 expands HTTP support for the Lifecycle Controller User Interface (LC-UI) to include export of server inventory, export of the Lifecycle Controller log, import and export of server profiles, and import of iDRAC with Lifecycle Controller licenses. Earlier, iDRAC with Lifecycle Controller firmware included HTTP support for firmware update only. These enhancements expand the options for network file share support with the Lifecycle Controller UI to include CIFS, NFS, and HTTP, simplifying Lifecycle Controller UI operations.

1.1.4 iDRAC REST with Redfish APIs

iDRAC9 includes enhancements to the iDRAC REST with Redfish APIs to include support for network file sharing via HTTP and HTTPS. In particular, the iDRAC REST APIs that support export, import preview and import of Server Configuration Profiles (SCP) have been enhanced to support accessing SCP files by using HTTP and HTTPS, in addition to existing support for CIFS and NFS file sharing.

For more information on using the iDRAC REST with Redfish APIs with HTTP/HTTPS, see the whitepaper *RESTful Server Configuration with iDRAC REST API*.

2 Extent of the changes

The HTTP and HTTPS changes extend beyond the interfaces and commands that perform data transfers.

- There are new attributes to support user proxy settings.
- There is a new attribute to support enabling/disabling HTTPS certificate verification.
- There is a facility to upload a certificate for HTTPS certificate verification.

2.1 RACADM Command Line Interface (CLI)

RACADM sub-commands allow HTTP and HTTPS to be given when entering a network location. Those that supported entering proxy settings with command line options still support them. For those sub-commands that don't have ways of entering proxy settings on the CLI, those values are taken from attributes. Also, the setting to validate or ignore the server certificate in the HTTPS case is also taken from an attribute.

When a proxy will be used with a RACADM sub-command be sure to set or verify the proxy attributes.

```
LifeCycleController.LCAttributes.UserProxyPassword
LifeCycleController.LCAttributes.UserProxyPort
LifeCycleController.LCAttributes.UserProxyServer
LifeCycleController.LCAttributes.UserProxyType
LifeCycleController.LCAttributes.UserProxyUserName
```

When HTTPS will be used with a RACADM sub-command be sure to set or verify the `LifeCycleController.LCAttributes.IgnoreCertWarning` attribute.

2.1.1 RACADM sub-commands with HTTP/HTTPS support

The following RACADM sub-commands provide support for HTTP/HTTPS:

- racadm update - firmware update using a single DUP or repository
- racadm get - import a server configuration profile
- racadm set - export a server configuration profile
- racadm lclog export - export the Lifecycle Log
- racadm hwinventory export - export the hardware inventory
- racadm inlettemphistory export - export the inlet temperature history
- racadm license export - export a license
- racadm license import - import a license
- racadm license replace - import a license to replace an existing one
- racadm autoupdatescheduler - perform firmware updates from a repository on a schedule
- racadm systemconfig backup - export a server profile backup image
- racadm systemconfig restore - import a server profile backup image
- racadm systemconfig backup - perform exports of backup images on a schedule
- racadm diagnostics export - export a diagnostics report
- racadm bioscert export - export BIOS Secure Boot Certificates. For more information see the whitepaper *Secure Boot Management In PowerEdge Servers* available on the support site.

- racadm bioscert import - import BIOS Secure Boot Certificates For more information see the whitepaper "Secure Boot Management In PowerEdge Servers".

2.2 WS-Man API support for HTTP/HTTPS

For many of the iDRAC9 WS-Man interfaces, the APIs have been enhanced by adding **ShareType** values for HTTP and HTTPS and by making **ShareName** optional. Keep in mind that not all methods use the same **ShareType** values; for examples **DM_LCService.ImportSystemConfigration** and **DCIM_SoftwareInstallationService.InstallfromRepository** use varying values for the same types.

DM_LCService.ImportSystemConfigration uses these values for **ShareType**:

- 0 = NFS
- 2 = CIFS
- 4 = LocalStore
- 5 = HTTP
- 6 = HTTPS

DCIM_SoftwareInstallationService.InstallfromRepository uses these values for **ShareType**:

- 0 = NFS
- 1 = FTP
- 2 = CIFS
- 3 = HTTP
- 4 = TFTP
- 6 = HTTPS

Check the description for each method to determine which values apply to that specific method.

2.2.1 WS-Man APIs and Profile with HTTP/HTTPS support

The following WS-Man APIs provide support for HTTP/HTTPS

- Exporting the Lifecycle Controller Log with ExportLCLog, ExportCompleteLCLog
- Exporting the Hardware Inventory with ExportHWInventory
- Exporting the Factory Shipped Configuration with ExportFactoryConfiguration
- Exporting the System Configuration Profile with ExportSystemConfiguration
- Exporting a backup server profile image with BackupImage, SetBackupSchedule
- Exporting a certificate with ExportCertificate
- Exporting a diagnostic report with ExportePSADiagnosticsResult
- Exporting a license with ExportLicenseToNetworkShare

- Import a System Configuration Profile with ImportSystemConfiguration, ImportSystemConfigurationPreview
- Import a server profile backup image with RestoreImage

- Install firmware update with InstallFromURI, InstallFromRepository, SetUpdateSchedule
- Import a license with ImportLicenseFromNetworkShare

2.3 Lifecycle Controller user interface

In this section, we provide the details about using the HTTP option from Lifecycle Controller on the 14th generation servers of Dell.

Note: iDRAC9 with Lifecycle Controller firmware version 3.00.00.00 does not support HTTPS. This white paper addresses only the use of HTTP in the Lifecycle Controller UI.

2.3.1 LC-UI support for HTTP

Prior to iDRAC with Lifecycle Controller firmware version 3.00.00.00 firmware, the Lifecycle Controller UI supported CIFS and NFS to exchange files with a network share based on the user selected file transfer method. Here are the features in Lifecycle Controller UI that support the HTTP file transfer method:

1. Import Features
   - Import/Restore Server profile image
   - Import Licenses
   - Unattended OS Deployment
2. Export Features
   - Lifecycle Controller Logs (LCL)
   - Hardware inventory
   - Factory shipped inventory
   - Server profile image
   - Tech Support Report

The following flow diagram shows the overall workflow of the HTTP method by using the Lifecycle Controller UI. The flow shows Import/Export mechanisms by using the IP network using HTTP as an option.

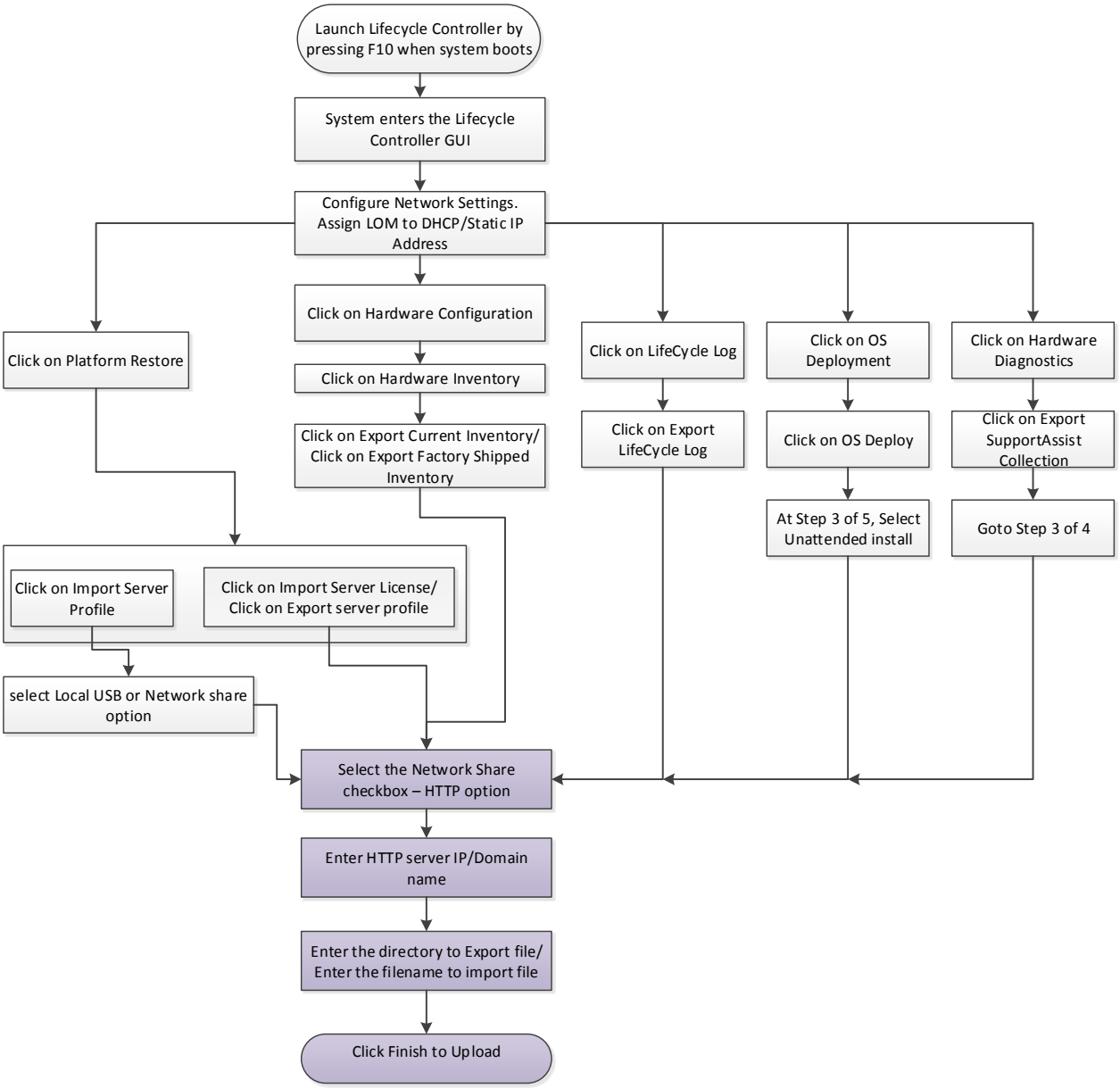

Figure 2 - Workflow of HTTP method via LC-UI interface

2.3.2 LC-UI HTTP deployment and configuration guide

2.3.2.1 HTTP method to export from Lifecycle Controller

Lifecycle Controller allows users to execute the following Export operations by using HTTP:

- Lifecycle Controller Logs (LCL)
- Hardware inventory
- Factory shipped inventory
- Server profile image
- Tech Support Report

Here is an example screen shot of the **Export Lifecycle Log** page,

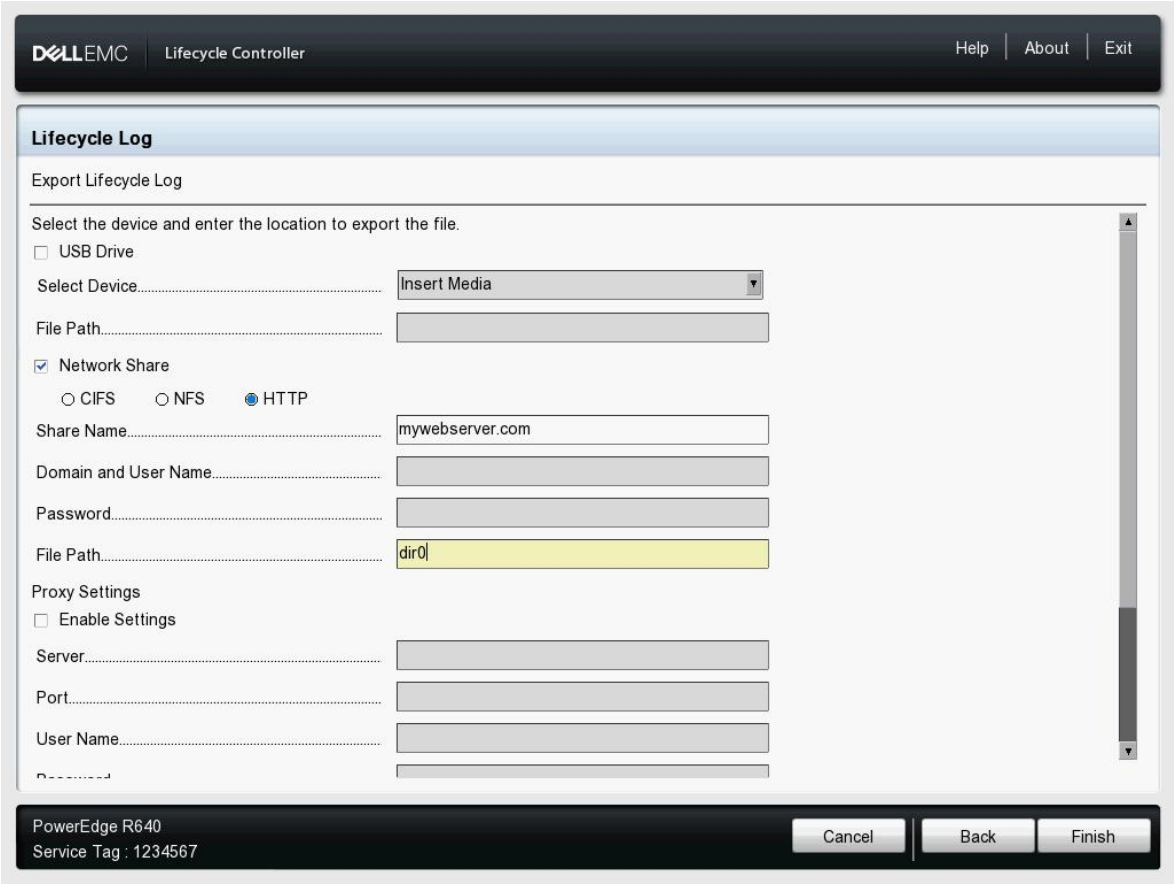

Figure 3 Example showing Export Lifecycle Log

1. Select the **Network Share** check box, and **HTTP** as the transfer method.
2. Enter the IP address or HTTP domain name in **Share Name** (For example, mywebserver.com or 10.1.1.10), and directory name or path in **File Path** (For example, dir0) to where the file must be exported
3. Click **Finish**.
   HTTP method does not require “user name” and “Password” Hence these field are grayed out for HTTP option.

Note: There is no option to browse through to the folder from Lifecycle Controller UI.

You can also have the proxy server options enabled, the details of proxy fields are:

- Server — the host name of the proxy server.
- Port — the port number of the proxy server.
- User Name — the user name required to access the proxy server.
- Password — the password required to access the proxy server.
- Type — the type of proxy server. Lifecycle Controller supports HTTP and SOCKS4 and SOCKS5 (for IPv6) proxy server types.

Note: The Lifecycle Controller web interface accesses the web server by using the default HTTP port (80). Ensure that the HTTP server is configured to the default port (80) to enable operations with the Lifecycle Controller UI. If the HTTP web server is configured to a port other than the default, Lifecycle Controller will not be able to access the web server.

2.3.2.2 HTTP method for import to Lifecycle Controller

Lifecycle Controller allows user to execute below Import operations by using HTTP:

- Import/Restore Server profile image
- Import License
- Unattended OS Deployment

Here is a sample screen shot of the **Import server License** page.

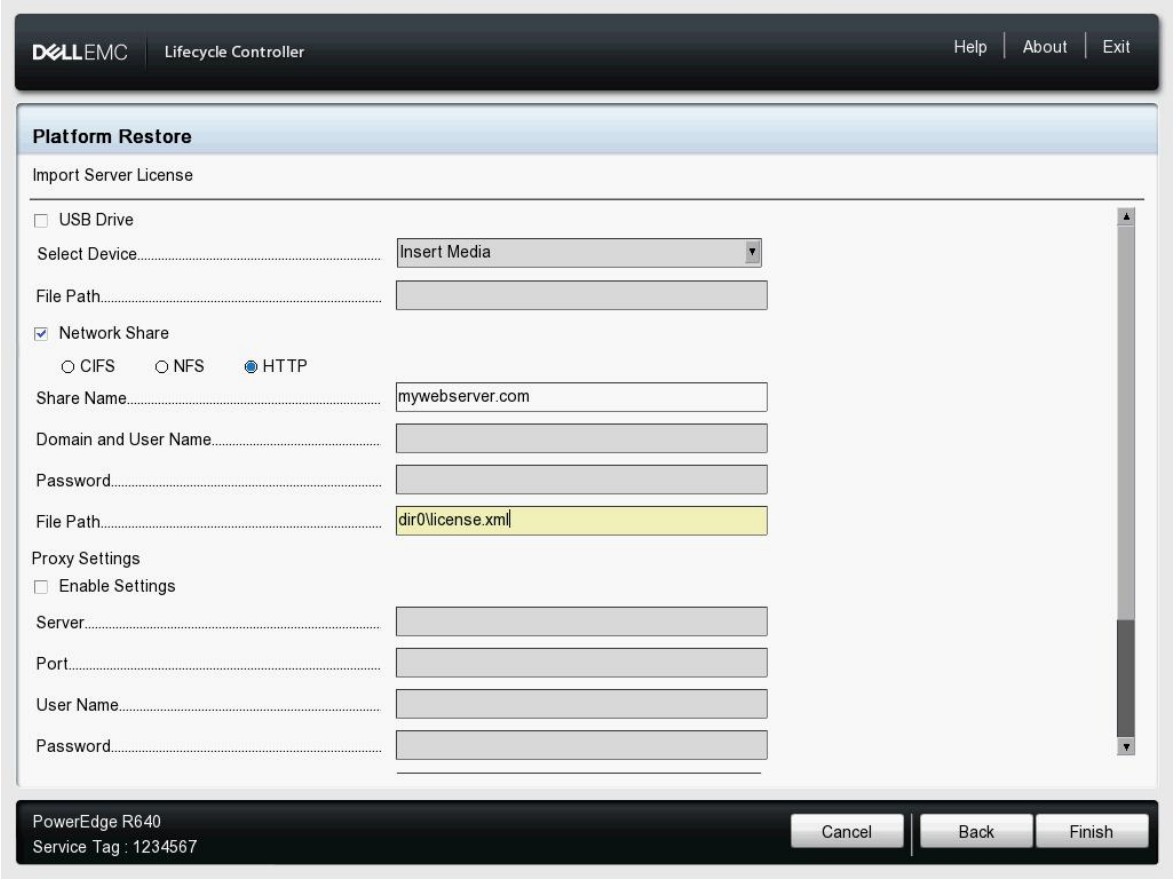

Figure 4 Example showing Import Server License

Type or select data in all the details about the HTTP share and its proxy details (if using proxy) and click **Finish**.

Note: Make sure to copy the License xml file in the HTTP repository. For information about the proxy option usage, see section 1.2.1 in this white paper.

2.3.2.3 Error scenarios and resolution

SWC0066: unable to connect to network share

Description: If user is using a server-name/IP address of HTTP:

- That is not reachable or
- The HTTP web server is not running on the specified IP/server-name or
- If HTTP web server is not configured to port 80 then the following message is displayed.

Solution: Make sure HTTP web server is reachable by pinging to the same, and that the HTTP web server is up and running, and check if the HTTP is configured on port 80.

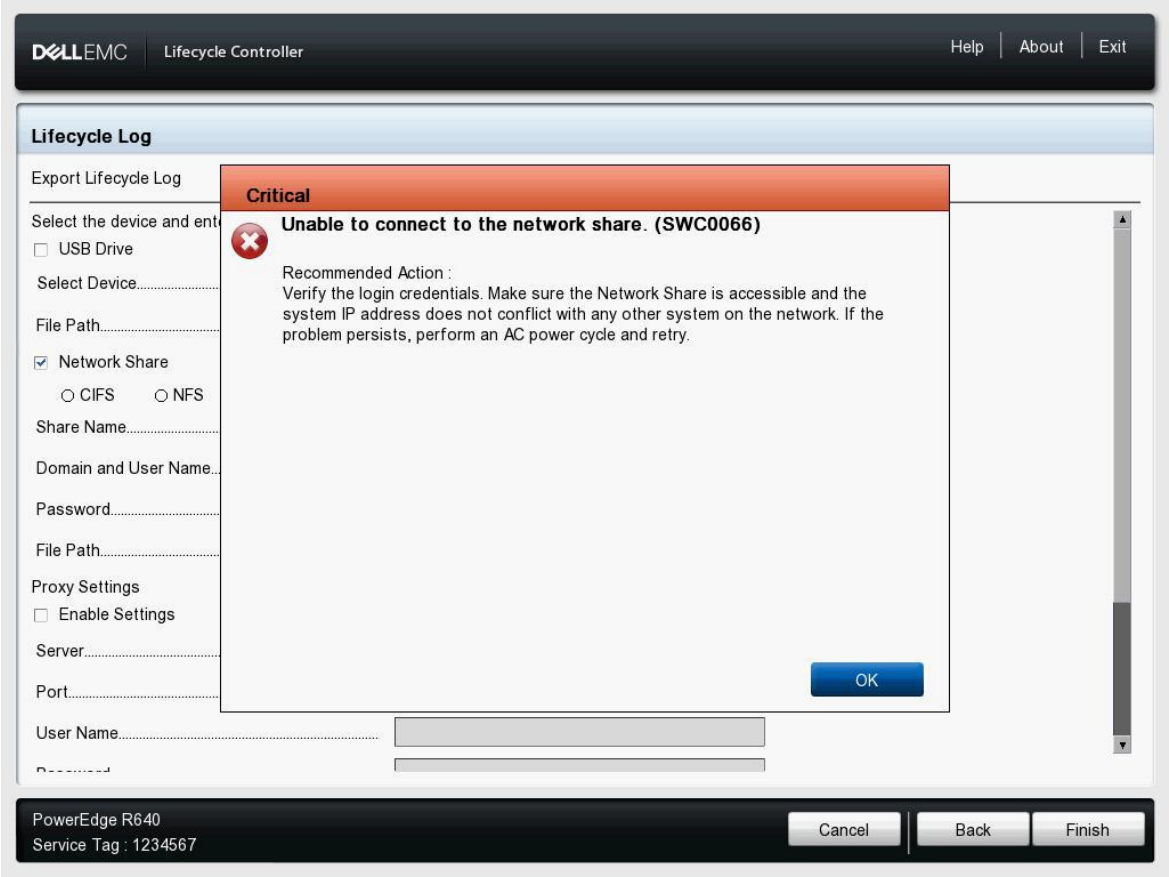

Figure 5 LC-UI unable to connect to the network share

SWC0037: unable to export the file to the network share

Description - If user tries to export/import a file to a folder that is not present on the HTTP repository, the message shows in the screen shot is displayed.

Solution – Make sure the folder/path to which the file is being exported is present and accessable (user has permission to access the folder).

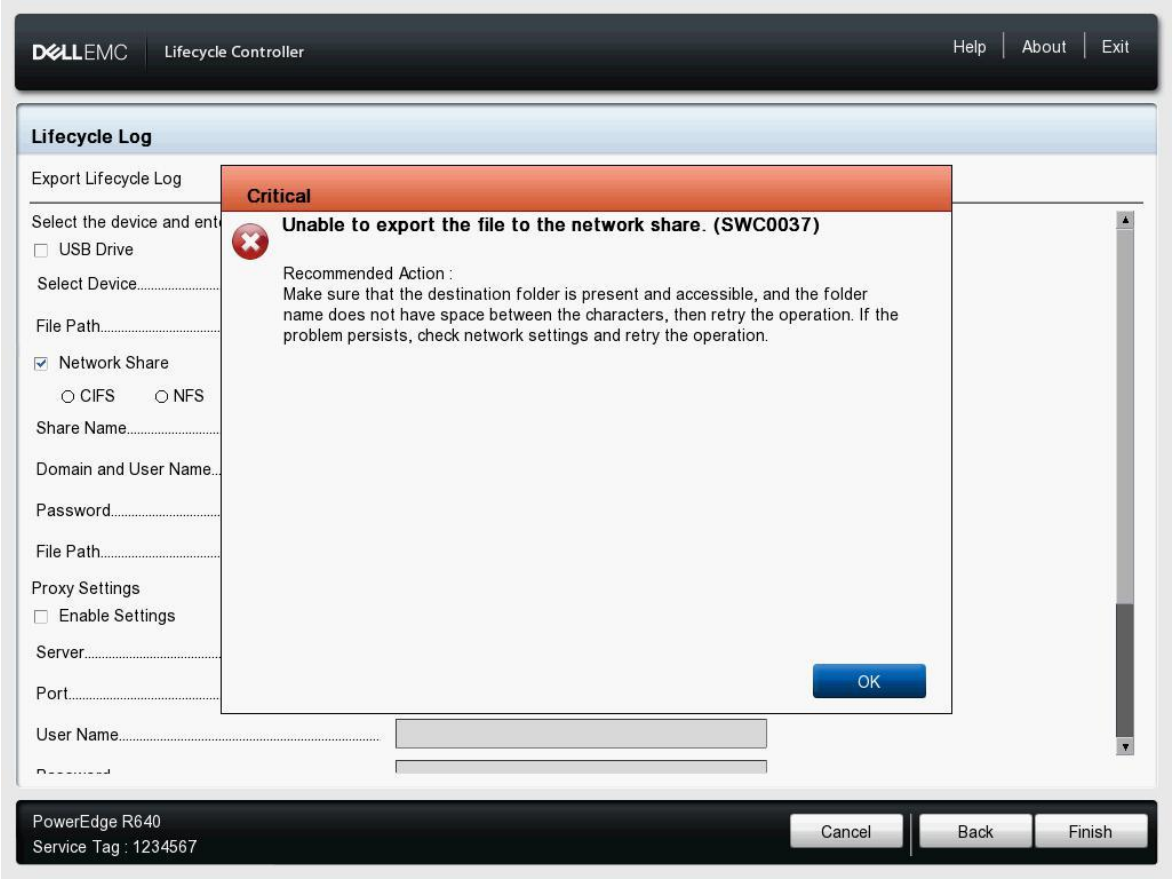

Figure 6 Lifecycle Controller UI unable export to the network share

SWC0057: unable to complete the operation

Description - If user specifies any share name followed by IP/server-name in the “Share Name” field then tries to export/import a file via HTTP then user will see this error message.

Solution – Make sure the “Share Name” field contains only the server-name/IP address of the HTTP server as expalined in section 1.2 of this white paper.

2.3.3 Best practices

Refer “Error Scenarios and resolution” under Section 1.2.3 for recommended actions in case of any error’s during Export or Import operations by using HTTP.

2.3.4 Lifecycle Controller user interface testing

Is	Is not
Tested and validated on all 14G servers, having iDRAC9 3.0.0.0.	Supported on 14G servers having iDRAC9 3.0.0.0 version are below.
Tested on Apache HTTP server 2.4.18 on Windows OS.	Tested on other than Apache HTTP server 2.4.18 on Windows OS.

2.3.5 Lifecycle Controller attributes

The following attributes are also used in interfaces other than the LC-UI. Attributes are available to allow values to be set when an interface is not able to set them itself. One set of these is for proxy settings.

```
LifeCycleController.LCAttributes.UserProxyPassword
LifeCycleController.LCAttributes.UserProxyPort
LifeCycleController.LCAttributes.UserProxyServer
LifeCycleController.LCAttributes.UserProxyType
LifeCycleController.LCAttributes.UserProxyUserName
```

These attributes are used with both HTTP and HTTPS.

The **UserProxyServer** is an important attribute. If it is not set then the other attributes cannot be used and the behavior will be as if none of them are set.

The **LifeCycleController.LCAttributes.IgnoreCertWarning** attribute is used only with HTTPS. If set to "On" then certificate warnings will be ignored. This is another way of saying HTTPS server certificate validation will not be done.

2.4 iDRAC9 Graphical User Interface (GUI)

The iDRAC GUI is available over HTTPS, as it has been. The addition of HTTP and HTTPS support refers to the way data is transferred over the network by operations initiated via the iDRAC GUI. The interface has

controls and fields to allow HTTP and HTTPS to be selected when entering a network location and allow proxy settings to be entered.

The following iDRAC9 GUI features support HTTP/HTTPS:

- Firmware Update and Scheduled Update
- Server Profile Backup, Scheduled Backup, and Import
- Server Configuration Profile export, preview import, and import
- Lifecycle Log Export
- Certificate upload
- License import and export

2.4.1 Certificate upload

There is an existing iDRAC facility to upload and store certificates for various uses. This has been expanded in iDRAC9 to allow the upload of the HTTPS server certificate. With the Apache server for example this will often be the ca.crt file found in /etc/pki/tls/certs. This allows the uploaded certificate to be compared with the HTTPS server certificate at the time of a data transfer to validate the identity of the server.

Note that this is not the certificate used with the iDRAC GUI or WS-Man for in-bound HTTPS connections to the iDRAC. It is the one used for out-bound connections from the iDRAC to an external HTTPS server. For example, using a web browser to connect to the iDRAC via HTTPS will use one certificate. When the iDRAC fetches a firmware update package from an HTTPS server, it uses this other different certificate.

2.4.2 Cryptography and Security

The addition of HTTPS adds complexity not present with other forms of transport like CIFS or NFS. New controls and facilities are necessary. Some previously unrelated iDRAC settings can now affect data transport.

The iDRAC does not have the facility to maintain an up-to-date certificate store for validating the identity of HTTPS servers. If a certificate store was included there would be no opportunity to update it with new or revoked certificates without an iDRAC firmware update. This required the addition of the facility to upload the server certificate for doing the server validation.

Using HTTPS without certificate validation may be useful to some. This required the addition of a way to turn validation off in the interfaces or with an attribute.

Part of the process of certificate validation is the comparison of the CN name in the certificate with the IP address or DNS name used to connect to the server. If a DNS name was used in the certificate then that name is required on the iDRAC. This requires DNS resolution be enabled on the iDRAC. DNS servers can be obtained via DHCP or can be set manually. These settings are off by default.

The implementation of HTTPS uses SSL, TLS and various cipher suites. It is possible for there to be a mismatch in what is supported. This means the iDRAC may not be able to communicate with a legacy HTTPS server. There is also a FIPS mode setting. Communication with a legacy HTTPS server may not be possible with FIPS turned on. This is normal and expected. A more up-to-date HTTPS server should be used.

Using HTTPS with a self-signed certificate and certificate validation turned off does have some usefulness. It is an incremental improvement over HTTP which transfers data in the clear. It prevents simple eavesdropping. It does not protect against main-in-the-middle attacks. But it does force an adversary to conduct a main-in-the-middle attack to get access to or alter the data. Using a certificate signed by a certificate authority with validation turned on is an improvement over both.

The iDRAC implementation does not support client authentication to the server.

For data connections outbound from the iDRAC (perfect), forward secrecy is preferred but not required.

3 Useful information for using HTTP and HTTPS

3.1 General file download info

HTTP and HTTPS web servers generally allow file downloads. But they may not allow or support unlimited file size and may only allow limited file types. The web server documentation should be consulted to adjust file size limits and file type permissions, securely, if needed.

3.2 General file upload info

HTTP and HTTPS web servers generally don't support file upload in the default configuration. The web server documentation should be consulted to enable file upload, securely, if needed.

3.3 Case sensitivity of URIs

With some HTTP and HTTPS servers it may appear that directory/folder and file names in URIs are case-insensitive. That is a property of the OS or file system being used on the server. (This isn't necessarily an error.) With different OSs and with different file systems the directory/folder and file names in URIs will be found as case sensitive as they really are. When in doubt and to avoid frustrating problems, match upper and lower case characters in all cases.

3.4 Encoding URIs

Sometimes URIs must contain special characters that interfere with proper parsing of the URI. These special characters are required to be encoded. This encoding should be done once when the URI is constructed from its component parts. When the user provides a URI to an interface, such as WS-Man or RACADM, the user must perform this encoding when needed. When the user provides the component parts, such as for many iDRAC GUI places, the user should not encode the component parts.

Passwords often have special characters because password rules often require them. This leads to URIs being inadvertently constructed without the required encoding. In general a http URI can look like this. (This is not an exact/robust definition. See RFC 3986.)

```
http://username:password@host:port/path/file
```

If the password for example were to contain special characters such as `pas@sw:ord` then it would need to be encoded like this.

```
http://username:pas%40sw%3aord@host:port/path/file
```

Inadvertently leaving it unencoded like this would cause the URI to be parsed incorrectly.

```
http://username:pas@sw:ord@host:port/path/file
```

There is also a special case with IPv6 numerical addresses where the: special character is not encoded. The IPv6 numerical address should be enclosed by using square brackets. This is not required with domain names that resolve to IPv6 addresses and not needed with IPv4 addresses.

```
http://username:pas%40sw%3aord@[fd76:6c:61:6e36:597f:7ec4:e1f:a97c]:port/p
ath/file
```

For more information, see RFC 3986 or its follow-on.

3.5 HTTPS Certificate generation

Certificates used with HTTPS are intended to be signed by a certificate authority. The certificate authorities publish public keys that allow the certificates to be validated. Software, such as web browsers, use a bundle of these public keys that is regularly updated to validate the certificates from web sites that are visited.

It is possible to create self-signed certificates, but software such as web browsers would not be able to validate them by using the bundle of public keys. Self-signed keys can protect against eavesdropping but not man-in-the-middle attacks.

3.5.1 DNS name matching

Note that the CN value specified in the Certificate Signing Request, CSR, must match the DNS name of the HTTPS server that will use the certificate. Otherwise the certificate may be rejected. That also means the DNS name must be used when referring to the server. Using an IP address to connect to the server will allow a connection to be made but the name comparison will fail and the certificate will be rejected. If a DNS name is not used with a server then the IP address can be used in the CN value. But that would require the server to always have that particular IP address via a static setting. Setting the server to use DHCP, with the possibility of getting one of a range of addresses, can cause a CN mismatch and the certificate will be rejected.

3.5.2 Using a Certificate Authority

These are some example commands to create a certificate on Linux for Apache. The basic steps will be the same for other OSs and servers but the commands and file names will be different. The basic steps are:

- Generate a private key
- Generate a Certificate Signing Request, CSR, from the key
- Send the CSR file to a certificate authority with the application or required info
- Receive the certificate signed by the authority
- Copy the files to the correct locations where they can be used
- Restart or reload the configuration of the web server

```
# Generate private key
openssl genrsa -out ca.key 2048

# Generate CSR
subj='/C=Norrath/ST=Starter Zone/L=Refuge Island/O=Blur/OU=Logo/CN=logo.blur.example.com'
openssl req -new -key ca.key -out ca.csr -subj "$subj"

# display and check the rsa key
openssl rsa -in ca.key -check -noout -text

# display the CSR
openssl req -text -noout -in ca.csr
```

```
echo ...

# display the certificate authority signed certificate
openssl x509 -in ca.crt -noout -text

# Copy the files to the correct locations
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

This is just an example. There are many types of keys that can be generated with different key lengths. The locations of the files will also change with Linux distribution and web server configuration.

3.5.3 Using a self-signed Certificate

These are some example commands to create a self-signed certificate on Linux for Apache. The basic steps will be the same for other OSs and servers but the commands and file names will be different. The basic steps are:

- Generate a private key
- Generate a Certificate Signing Request, CSR, from the key
- Generate a self-signed certificate
- Copy the files to the correct locations where they can be used
- Restart or reload the configuration of the web server

```
# Generate private key
openssl genrsa -out ca.key 2048

# Generate CSR
subj='/C=US/ST=Arizona/L=Monument Valley/O=ACME/OU=Anvil Division/CN=anvil.acme.example.com'
openssl req -new -key ca.key -out ca.csr -subj "$subj"

# Generate Self-Signed Certificate
openssl x509 -req -days 3652 -in ca.csr -signkey ca.key -out ca.crt

# display and check the rsa key
openssl rsa -in ca.key -check -noout -text

# display the CSR
openssl req -text -noout -in ca.csr

# display the self-signed certificate
openssl x509 -in ca.crt -noout -text

# Copy the files to the correct locations
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

This is just an example. There are many types of keys that can be generated with different key lengths. The locations of the files will also change with Linux distribution and web server configuration.

3.6 Apache server info

This sections contains info that may help by using an Apache HTTP and HTTPS server. This info may go out of date as time passes since publication. This info was gathered when working with Server: Apache/2.2.15 and PHP version 5.3.3.

3.6.1 Uploading files to Apache

Uploading files to an Apache HTTP and HTTPS server requires a PUT script. The script name comes from the protocol command that is used. This ability to upload a file is used when transferring data from the iDRAC to the Apache server via HTTP and HTTPS. This is useful for:

- Exporting the Lifecycle Controller Log
- Exporting the Hardware Inventory
- Exporting the Factory Shipped Configuration
- Exporting the System Configuration
- Exporting a backup image
- Exporting a certificate
- Exporting a diagnostic report
- Exporting a license

If a given use case does not require exporting data from the iDRAC then it is not necessary to configure or use this type of script. Transfers from the server to the iDRAC do not use it.

The Apache server does not support PUT (upload) requests by default. It must be configured to pass the PUT requests to a script to process the request. There is no native, non-script support for PUT requests in Apache.

An example `put.php` script is shown in Appendix A. It requires modification before it can be used. It was written with the goal of allowing file upload safely, but history shows that software that was thought safe is sometimes later found unsafe. If a variation of the example script is used then the user should be prepared to update it as issues are discovered.

This is an example configuration file fragment for enabling `put.php`. The script could be placed in the configured cgi-bin directory, but keep in mind it is not a CGI script. In that case, the line "Script PUT /scripts/put.php" would be something like "Script PUT /cgi-bin/put.php" and the ScriptAlias and Directory settings for "/some_safe_directory/scripts/" wouldn't be needed. As always, be sure to modify it as needed to be secure.

```
<Directory "/web_server_root">
    Order allow,deny
    Allow from all
    <Limit PUT DELETE>
        Order allow,deny
        Allow from all
    </Limit>
    <LimitExcept PUT DELETE>
        Order deny,allow
        Deny from all
    </LimitExcept>
```

```
    Script PUT /scripts/put.php
</Directory>
ScriptAlias /scripts/ "/some_safe_directory/scripts/"
<Directory "/some_safe_directory/scripts">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

PHP may need to be installed. Installing Apache does not necessarily pull in the PHP package. If this is the first PHP script being configured for a particular server and it isn't working, this is a quick first thing to check.

- The Apache server does not support authentication in the default configuration. It must be configured with user credentials.
- Credentials given for HTTP connections are not secure. Use HTTPS to use server credentials securely.

3.7 Microsoft-IIS server info

This sections contains info that may help with using a Microsoft-IIS HTTP and HTTPS server. This info may go out of date as time passes since publication. This info was gathered when working with Server: Microsoft-IIS/8.5, X-AspNet-Version: 4.0.30319t.

3.7.1 Downloading files from Microsoft-IIS server

The default server configuration has some limits that may need to be adjusted. There are limits on file types.

3.7.1.1 File type limitation

The server matches file name extensions to MIME types. With the default configuration, files that have extensions that are not mapped to a MIME type are not allowed to be downloaded. If this is encountered the file name can be changed to a known type which could then be downloaded. For more information see https://support.microsoft.com/en-us/help/326965/iis-6.0-does-not-serve-unknown-mime-types.

3.7.2 Uploading files to Microsoft-IIS server

The default server configuration has some limits that may need to be adjusted. There is a limit on file size. Folders are not created automatically.

3.7.2.1 File size limitation

The value of the default upload file size limit varies with server version. The limit may be adjusted by changing the "maxAllowedContentLength" value in various places or using the IIS Manager to modify "Features View" -> "Request Filtering" -> "Edit Feature Settings" -> "Request Limits" -> "Maximum allowed content length (Bytes):". This info may vary or be out of date. Please see the documentation for the server version being used for more authoritative and up-to-date info. For more information see **https://www.iis.net/configreference/system.webserver/security/requestfiltering/requestlimits**.

3.7.2.2 Folder creation

When a file is uploaded, the destination Folder must already exist. Else, the upload will fail. The user must create the folder in the backing filesystem using some other method than through the PUT method.

3.8 General proxy information

3.8.1 Using HTTP with a proxy

When using credentials with a proxy to connect to an HTTP server to send or upload a file, without using tunneling, there may be a performance impact. Because of the sequence used for the credentials the data file payload may be sent two or three times. This could cause the transfer to take two to three times longer than when not using proxy credentials. This does not apply to receiving or downloading a file. It also does not apply when proxy credentials are not used.

3.8.2 Using HTTPS with a proxy

When using HTTPS with a proxy the connection between the iDRAC and the proxy is not as secure as the connection between the iDRAC and the HTTPS server. The connection between the iDRAC and the HTTPS server is encrypted and credentials used to log into the server (if any) are carried over the encrypted connection. The connection between the iDRAC and the proxy is not encrypted. The credentials used to log into the proxy (if any) are transferred before the encrypted connection is started. Because of this the credentials used to log into the proxy should not be the same credentials used to log into the server. That way if the proxy credentials are compromised it means the HTTPS server credentials are not also compromised.

3.9 Choice of proxy

During development and test many different proxy packages were tested running on various Linux and Windows releases. Each had different capabilities and could be configured and customized to varying degrees. It appears that no two operated alike. It may be necessary to choose different proxy packages for different use cases. The two packages listed below are intended as a representative sample of the kinds of variation that may be found.

3.9.1 Squid proxy info

- The Squid proxy does not support HTTP PUT requests. In other words, uploading a file via HTTP through a squid proxy fails.
- The Squid proxy does support PUT requests for HTTPS connections because they are tunneled. This only applies when tunneling for HTTPS is enabled (which it is by default).
- The Squid proxy does not support authentication in the default configuration. It must be configured with user credentials.
- Proxy credentials are not secure in both the HTTP and HTTPS cases. There is no way to have secure proxy credentials through a proxy because the client-proxy connection is not encrypted. Do not use the same credentials for the server and the proxy.

3.9.2 Tinyproxy proxy info

- The Tinyproxy proxy does not support HTTP PUT requests when authentication is required. The problem has to do with the need to upload data two or more times when handling "authentication required" responses.
- The Tinyproxy proxy does support PUT requests for HTTPS connections because they are tunneled. This only applies when tunneling for HTTPS is enabled (which it is by default).
- The Tinyproxy proxy does not support authentication in the default configuration. It must be configured with user credentials.
- Proxy credentials are not secure in both the HTTP and HTTPS cases. There is no way to have secure proxy credentials through a proxy because the client-proxy connection is not encrypted. Do not use the same credentials for the server and the proxy.

A An example PUT script

A PUT script allows a file to be uploaded to an Apache HTTP and HTTPS server. The name comes from the protocol command that is used. This ability to upload a file is used when transferring data from the iDRAC to the Apache server via HTTP and HTTPS. This is useful for:

- Exporting the Lifecycle Controller Log
- Exporting the Hardware Inventory
- Exporting the Factory Shipped Configuration
- Exporting the System Configuration
- Exporting a backup image
- Exporting a certificate
- Exporting a diagnostic report
- Exporting a license

If a given use case does not require exporting data from the iDRAC then it is not necessary to configure or use this type of script. Transfers from the server to the iDRAC do not use it.

This is an example put.php script for uploading files to an Apache HTTP and HTTPS server. This is an example and not a supported software product. It requires modification before it can be used. It was written with the goal of allowing file upload safely, but history shows that software that was thought safe is sometimes later found unsafe. If a variation of the example script is used then the user should be prepared to update it as issues are discovered.

This is a script to handle PUT requests to the server. It is not a CGI script. It looks a lot like one and some of the configuration is the same. It can configured to be in the same place as CGI scripts. But it doesn't operate the same.

This is the file 'put.php'.

```
<?php

// This script is an example.
// Please adjust and secure it for your environment.

// Directory mode. If this script creates a directory it will
// be given this file mode. Files and directories will be owned
// by the user and group of the server process.
$new_directory_mode=0770;

// File mode. If this script creates a file it will
// be given this file mode. Files and directories will be owned
// by the user and group of the server process.
$new_file_mode=0640;

// Buffer size. This is how much data will be transferred from
// input to the destination file per chunk.
$buffer_size=1024;

$no_overwrite=false;
```

```
// Set this to true to disable overwriting files that already exist.
//$no_overwrite=true;

ini_set("allow_url_fopen",false);

// PUT root
// Set $putroot to the directory under which destination
// files will be written.
// Use this if you want files to go under the document root
// defined by the server config file.
//$putroot=$_SERVER['DOCUMENT_ROOT'];
$putroot=$_SERVER['DOCUMENT_ROOT'];
// Use this if you want files to go under a subdirectory of
// the document root defined by the server config file. In
// this case they will go under the "pub" directory under the
// document root.  This makes it so that nothing outside that
// subdirectory can be overwritten by a PUT request.
//$putroot=$_SERVER['DOCUMENT_ROOT']."/pub/";

// This will restrict the location of destination files. The translated
// path part of the URI must begin with this string or the
// request will be rejected. This makes it so that nothing outside
// the given path can be overwritten by a PUT request.
// This is relative to the DOCUMENT_ROOT.  If $putroot is not
// set to the DOCUMENT_ROOT then this value will have to take
// that into account.
//$restrict_to="/";
//$restrict_to="/pub/";
$restrict_to="/pub/";

// Generate a document with the response code, short
// message, and verbose message.
function gen_doc($code,$message,$verbose)
{
    $message_encoded=htmlspecialchars($message);
    $verbose_encoded=htmlspecialchars($verbose);
    echo '<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">'."\n";
    echo "<http>\n";
    echo "<head>\n";
    echo "<title>".$code." ".$message_encoded."</title>\n";
    echo "</head>\n";
    echo "<body>\n";
    echo "<h1>".$code." ".$message_encoded."</h1>\n";
    echo "<p>".$verbose_encoded."</p>\n";
    echo "</body>\n";
    echo "</http>\n";
}

// Output a header line with the response code and
// short message.
// Generate a document with the response code, short
// message, and verbose message.
function header_doc($code,$message,$verbose)
{
    script_debug("HTTP/1.0 ".$code." ".$message);
    header("HTTP/1.0 ".$code." ".$message);
    header("DebugVerbose: ".$verbose);
    gen_doc($code,$message,$verbose);
```

```
}

// Output just the header line with the response code
// and short message, then exit.
function header_exit($code,$message,$exit_value)
{
    script_debug("HTTP/1.0 ".$code." ".$message);
    header("HTTP/1.0 ".$code." ".$message);
    exit($exit_value);
}

// Output the header line with the response code and
// short message.
// Generate a document with the response code, short
// message, and verbose message.
// Then exit.
function header_doc_exit($code,$message,$verbose,$exit_value)
{
    header_doc($code,$message,$verbose);
    exit($exit_value);
}

// This script is an example.
// Please adjust and secure it for your environment.
function example_code()
{
    header_doc_exit(418,"This script is an example. Please adjust and secure it for your
environment.","",1);
}

function script_debug($message)
{
    global $dboutFP;
    if( ! $dboutFP )
    {
        return;
    }
    $ending="\n";
    $len=strlen($message);
    if( $len >= 1 && substr_compare($message,$ending,-1) === 0 )
    {
        $ending="";
    }
    fwrite($dboutFP,$message.$ending);
    fflush($dboutFP);
}

function script_debug_variables()
{
    foreach($_SERVER as $key => $value)
    {
       script_debug($key." ".$value."\n");
    }
}

function copy_input_to_file($outname)
{
    global $new_file_mode, $buffer_size;
```

```
    $inFP=@fopen("php://input","rb");
    if( ! $inFP )
    {
        header_doc_exit(500,"Internal script error.","Unable to open the php input
'php://input'.",1);
    }
    $outFP = @fopen($outname, "wb" );
    if( ! $outFP )
    {
        header_doc_exit(403,"Unable to open.","Unable to open the destination file. The
filesystem or directory may be readonly. The filesystem may be out of space or inodes.
filename=[".$outname."]",1);
    }
    chmod($outname,$new_file_mode);
    $waswritten=0;
    while(!feof($inFP))
    {
        $data=fread($inFP,$buffer_size);
        if( $data === false ) {
            // The read failed.
            header_doc_exit(403,"Unable to read.","Unable to read the input.",1);
        }
        $rc=fwrite($outFP,$data);
        if( $rc === false ) {
            // The write failed.
            header_doc_exit(403,"Unable to write.","Unable to write to the destination file.
Opening the file was ok. The filesystem may be out of space. filename=[".$outname."]",1);
        }
        $l=strlen($data);
        if( $l != $rc ){
            // write length mismatch
            header_doc_exit(403,"Failed write length.","Unable to complete a write to the
destination file. Opening the file was ok. The filesystem may be out of space.
filename=[".$outname."]",1);
        }
        $waswritten += $rc;
    }
    fclose($inFP);
    fclose($outFP);
    return $waswritten;
}

example_code();

$dboutFP=false;
// This line opens a debug output file under the PUT root for
// debugging this script.  Comment-out to disable debug.
//$dboutFP = @fopen($putroot."/debugout.txt","wb");
$dboutFP = @fopen($putroot."/debugout.txt","wb");

script_debug_variables();

// REQUEST_URI is percent encoded
// PATH_INFO has been decoded
// PATH_TRANSLATED is decoded and could be used instead of PATH_INFO if
//     $putroot was set to DOCUMENT_ROOT.
// $fullpath is the full path name to the destination file.
```

```
//$fullpath=$_SERVER['PATH_TRANSLATED'];
//$fullpath=$putroot.$_SERVER['PATH_INFO'];
$fullpath=$putroot.$_SERVER['PATH_INFO'];

// Clean out any multiple / in the path to get ready for the comparison.
script_debug("fullpath ".$fullpath."\n");
$fullpath=preg_replace('/\/\/+/','/',$fullpath);
script_debug("fullpath ".$fullpath."\n");

// Get the full restriction path relative to the host filesystem.
$full_restrict_to=$_SERVER['DOCUMENT_ROOT'].$restrict_to;

// Clean out any multiple / in the path to get ready for the comparison.
script_debug("full_restrict_to ".$full_restrict_to."\n");
$full_restrict_to=preg_replace('/\/\/+/','/',$full_restrict_to);
script_debug("full_restrict_to ".$full_restrict_to."\n");

if( substr_compare($fullpath,$full_restrict_to,0,strlen($full_restrict_to)) === 0 )
{
    script_debug("The desitnation file path is inside the restricted path.\n");
}
else
{
    header_doc_exit(403,"Forbidden file destination.","The destination file path is outside the
allowed destination path. filepath=[".$fullpath."] restrictedto=[".$full_restrict_to."]",1);
}

// Get the name of the file without any of the directory path.
$filebase=basename($fullpath);
// Get the directory path of the file without the name.
$filedir=dirname($fullpath);

script_debug("filebase ".$filebase."\n");
script_debug("filedir ".$filedir."\n");

// See if any directories need to be created before we are able
// to write the destination file.
if( is_dir($filedir) )
{
    script_debug("The directory already exists.");
}
else
{
    script_debug("The directory does not exist.\n");
    if( ! mkdir($filedir,$new_directory_mode,true) )
    {
        // The mkdir failed.
        script_debug("mkdir failed\n");
        header_doc_exit(403,"Unable to make a directory.","Unable to make a directory needed in
the destination path. The filesystem or parent directory may be readonly. The filesystem may be
out of space. dir=[".$filedir."]",1);
    }
}

// Check for an existing file if we don't want to allow any overwrites.
if( is_file($fullpath) )
{
    script_debug("The destination file already exists.");
```

```
    if( $no_overwrite )
    {
        header_doc_exit(403,"File exists.","The destination file already exists. The PUT script
has been configured to not allow overwrites. filepath=[".$fullpath."]",1);
    }
}

// Copy the PUT data to the destination file.
$waswritten=copy_input_to_file($fullpath);

// Indicate success.
header_doc("200","Success","The PUT operation was successful.");

if( $dboutFP )
{
    fclose($dboutFP);
    $dboutFP=false;
}

// vim:expandtab:tabstop=4:shiftwidth=4
?>
```

B Configuration details

Table 1 Component table

Component	Description
Firmware version	LC 3.0.0.0 , iDRAC9 3.00.00.00
Server	All Dell PowerEdge 14G servers
HTTP Server	Apache 2.4.18
PHP 7	PHP language used for server-side web development.

- For Apache HTTP server configurations, visit:
  - https://danielarancibia.wordpress.com/2015/09/27/installing-apache-2-4-and-php-7-for-development-on-windows/
- You can also refer to some of the below YouTube links to install Apache on windows,
  - https://www.youtube.com/watch?v=fYrBLw_A8zU
  - http://www.wikihow.com/Install-the-Apache-Web-Server-on-a-Windows-PC

Note: The "**Listen**" string to configure port of Apache under "httpd.conf" file should be set to 80 to work with LCUI.

C Glossary

Term	Meaning
HTTP	Hyper Text Transfer protocol
NFS	Network File System
CIFS	Common Internet File System

D Technical support and resources

- Dell.com/support is focused on meeting customer needs with proven services and support.
- Dell TechCenter is an online technical community where IT professionals have access to numerous resources for Dell EMC software, hardware and services.
- Storage Solutions Technical Documents on Dell TechCenter provide expertise that helps to ensure customer success on Dell EMC Storage platforms.