Live Endpoint Response Client Documentation

Release 1.0.0

Cole Robinette & Sean McFeely

Jul 14, 2021

Contents:

The Live Endpoint Resposne Client (LERC) provides a service for resumable file transfer from client to server to analyst/user. Additionally, LERC acts as a Remote Access Solution with the following commands:

CHAPTER 1

Major Features

- Fast data transfer via chuncked streaming
- Bi-directional file transfers automatically resume after broken connections
- Run shell commands on clients and stream back results
- Installed as a service with a MSI installer
- Capability to sign the MSI and EXE you build
- Client can be set to trust a custom server signing cert
- All clients are verified by the server by a certificate you specify
- Control API to interact with the server and control the clients
- Clients will uninstall and delete themselves when issued a "quit" command
- Server will only accept control commands from a verified certificate you provide

1.1 The LERC Protocol

Each endpoint fetches the current command from the server by sending a GET request to /fetch. The machine name of the endpoint is passed in the query string variable "host" and an integer is passed representing the company/organization/group the host belogs to in the variable "company". Below is an example fetch request uri.

```
https://your-server/fetch?host=WIN1234&company=0
```

The server then returns a JSON string containing the commands for the host to execute. The supported commands are:

- Run
- Sleep
- Download
- Upload

- Quit

1.2 LERC Commands

1.2.1 Sleep

The sleep command tells the endpoint to do nothing for some number of seconds. Below is an example sleep command which tells the endpoint to do nothing for 30 minutes.

```
{ "operation":"sleep", "seconds":1800 }
```

Parameters

operation The operation to perform.

seconds Number of seconds to wait before contacting the server again.

1.2.2 Run

The run command tells the endpoint to execute a shell command. Below is an example run command which tells the endpoint to execute "echo hello world"

```
{ "operation":"run", "id":"1234", "command":"echo hello world" }
```

Parameters

operation The operation to perform.

command The shell command to execute.

Result

The endpoint will begin executing the command and streaming the output back to the server in a POST request to /pipe. The machine name and command id are sent via the query string variables "host" and "id".

```
https://your-server/pipe?host=WIN1234&id=1234
```

1.2.3 Download

The download command is for sending files to endpoints.

```
{ "operation":"download", "id":"1234", "path":"c:\\HelloWorld.txt" }
```

Parameters

operation Ther operation to perform

id The command id

path The path on the endpoint to write the data

Result

The endpoint sends a GET request to /download. The query string will contain the variables "host", "id" and "position". The host variable contains the machine name. The id variable is the command id. The position variable tells the server how much of the file is already downloaded. The server should then start streaming data to the client.

```
https://your-server/download?host=WIN1234&id=1234&position=0
```

1.2.4 Upload

The upload command is for sending files from endpoints to the server.

```
{ "operation":"upload", "id":"1234", "path":"c:\\HelloWorld.txt", "position":0
}
```

Parameters

operation The operation to perform

id The command id

path The path on the endpoint to read the data from

position The position in the file to start reading data. This is used to resume upload commands

Result

The endpoint sends a POST request to /upload and begins streaming data to the server. The query string contains the host, id and size variables. The host variable is the machine name. The id variable is the command id. The size variable is the size of the target file.

```
https://your-server/upload?host=WIN1234&id=1234
```

1.2.5 Quit

The Quit command tells the client to uninstall itself from the endpoint.

```
{ "operation":"quit" }
```

1.2.6 Errors

If LERC encounters an unexpected error while executing a command it will send a POST request to /error. The query string will contain the host and id variables. The host variable is the machine name and the id variable is the command id. The post data will contain the error message.

```
https://your-server/error?host=WIN1234&id=1234
```

1.3 Server Installation

1. Start with a clean Ubuntu 18 LTS Server install
2. Git the lerc files in */opt/*:

```
cd /opt && sudo -E git clone https://github.com/IntegralDefense/lerc.git
```

3. Create lerc user:

```
sudo adduser lerc
```

4. Give the lerc user sudo:

```
sudo adduser lerc sudo
```

5. Give lerc full access to */opt/lerc*:

```
cd /opt && sudo chown -R lerc:lerc lerc
```

6. Become the lerc user:

```
sudo su lerc
```

7. Run the install script:

```
cd /opt/lerc/lerc_server && installer/source_install
```

8. Chose to either change the default LERC Client group/company name or use the 'default' name:

```
Do you wish specify the default client group/company name? The default is 'default
↪'.
1) Yes
2) No
```

9. Either supply a password for the MySQL DB or let the install script generate one:

```
Generate random password for LERC DB user or supply your own?
1) Generate
2) Supply
```

10. Decide how you want to run the LERC server:

```
Running the LERC Server with one interface or two? Choose one if you don't know␣
↪what you want.
 -- One Interface: Both the LERC clients and analysts (control API) will access␣
↪this server on the same interface.
 -- Two Interfaces: LERC Clients and analysts (control API) will access their␣
↪features via different interfaces.
Choose which interface configuration you want:
1) One
2) Two
```

After that completes your LERC server should be running and ready. As part of the installation, LERC Client and LERC Control certificates were generated and placed in the following locations.

Client:

```
/opt/lerc/lerc_server/ssl/client/
```

Control:

```
/opt/lerc/lerc_server/ssl/admin/
```

These certificate files, and the */opt/lerc/lerc_server/ssl/ca-chain.cert.pem* will be needed to configure the LERC Control and LERC Clients.

As an additional security measure, you can safely remove the *lerc* user from the sudoers file after installation.

1.4 LERClient Setup

If you want to build your own client version, feel free. Otherwise, there is a LERC.zip file included in the project repo that contains everything you need to start.

Download the LERC.zip file and extract it. Then, copy the client certificats that were generated by the server installation into that same directory. Next, edit the config.txt file so that the 'serverurls' variable reflects your LERC server. For example:

```
serverurls: https://3.214.24.217/
```

Run lerc.exe and *tail -f /opt/lerc/lerc_server/logs/server.log* to see the client fetch.

1.5 LERC Control

A local source install is made on the system that the LERC server is installed on. For lerc-control on other systesms, use the certificates that are generated by your LERC server install or use your own certs.

The LERC control library is a python wrapper around the server API that provides several ulitlites and functions for controling and issuing commands to clients. A `lerc_ui` script is included if installed with pip3 that provides a powerful command line user interface to this library.

1.5.1 Setup

Use pip3 to install lerc-control:

```
pip3 install lerc-control
```

Note that you will need to have a working LERC Server and workign clients to use LERC Control.

1.5.2 LERC API Library

The LERC API module is the foundation for interacting with clients. The LERC Control library is built around it.

1.5.3 Control Library

The LERC Control library used the LERC API to perform live response funtions, such as performing scripted routines, as well as more complex collections and remediations.

Structure:

```
/lerc_control
  __init__.py
  scripted.py
  collect.py
```

(continues on next page)

(continued from previous page)

```
  remediate.py
  helpers.py
```

Scripted

The scripted module should only contain classes and functions for running or related to scripted routines.

Collect

All Live Response collection related classes and function belong in the collect module.

Remediate

All Live Response remediation related classes and functions belong in the remediate module.

Helpers

Global helper and general functions and classes belong in this module.

1.5.4 LERC User Interface

The `lerc_ui` or `lerc_ui.py` script can be used to perform several automated functions. Below is a description of the commands you can run with it:

```
$ lerc_ui -h
usage: lerc_ui.py [-h] [-e ENVIRONMENT] [-d] [-c CHECK] [-r RESUME] [-g GET]
                  {query,run,upload,download,quit,collect,contain,script} ...

User interface to the LERC control server

positional arguments:
  {query,run,upload,download,quit,collect,contain,script}
    query               Query the LERC Server
    run                 Run a shell command on the host.
    upload              Upload a file from the client to the server
    download            Download a file from the server to the client
    quit                tell the client to uninstall itself
    collect             Default (no arguments): perform a full lr.exe
                        collection
    contain             Contain an infected host
    script              run a scripted routine on this lerc.

optional arguments:
  -h, --help            show this help message and exit
  -e ENVIRONMENT, --environment ENVIRONMENT
                        specify an environment to work with. Default='default'
  -d, --debug           set logging to DEBUG
  -c CHECK, --check CHECK
                        check on a specific command id
  -r RESUME, --resume RESUME
```

(continues on next page)

(continued from previous page)

```
                        resume a pending command id
  -g GET, --get GET     get results for a command id
```

Examples

Killing a process and deleting dir

Below, using `lerc_ui.py` to tell the client on host "WIN1234" to run a shell command that will kill *360bdoctor.exe*, change director to the directory where the application is installed, delete the contents of that directory, and then print the directory contents. The result of this command should return an emptry directory.

```
$ lerc_ui.py run WIN1234 'taskkill /IM 360bdoctor.exe /F && cd
↪"C:\Users\bond007\AppData\Roaming\360se6\Application\" && del /S /F /Q
↪"C:\Users\bond007\AppData\Roaming\360se6\Application\*" && dir'
```

Querying

The server supports a very basic query language. Query fields are only ANDed together and negation is supported by placing a '-', '!', or 'NOT ' in directly in front of the field to be negated. Note, the `-rc` option will explicitly return commands is set, else commands are only returned if a 'cmd_*' field is specified in the query.

Available Fields:

Field Description ============== == cmd_status The status of a command: pending,preparing,complete,error,unknown,started cmd_id The ID of a specific command company A company/group name client_status A LERC status: busy,online,offline,unknown,uninstalled version The LERC version string hostname The hostname of a client company_id Specify a company/group ID client_id Specify a LERC by ID operation A Command operation type: upload,run,download,quit

Query for a specific host:

```
$ lerc_ui.py query 'hostname:w7gotchapc'

Client Results:

ID     Hostname              Status       Version   Sleep Cycle  Install Date
↪ Last Activity         Company ID
=====  ====================  ===========  ========  ===========  ====================
↪ ====================  ==========
14     W7GOTCHAPC            OFFLINE      1.0.0.0   60           2018-12-12 14:19:18
↪ 2018-12-14 14:40:56   0
Total Client Results:1
```

Not Run commands that have errored for this host, which is not online:

```
$ lerc_ui.py query 'hostname:w7gotchapc -client_status:online -operation:run cmd_
↪status:error'

Client Results:

ID     Hostname              Status       Version   Sleep Cycle  Install Date
↪ Last Activity         Company ID
=====  ====================  ===========  ========  ===========  ====================
↪ ====================  ==========
```

(continues on next page)

(continued from previous page)

```
14      W7GOTCHAPC              OFFLINE       1.0.0.0    60           2018-12-12 14:19:18
↪ 2018-12-14 14:40:56   0
Total Client Results:1


Command Results:

ID          Client ID  Hostname              Operation    Status
=========  =========  ====================  ===========  =========
320         14         w7gotchapc            DOWNLOAD     ERROR
322         14         w7gotchapc            DOWNLOAD     ERROR
377         14         w7gotchapc            UPLOAD       ERROR
609         14         w7gotchapc            UPLOAD       ERROR
696         14         w7gotchapc            UPLOAD       ERROR
807         14         w7gotchapc            UPLOAD       ERROR
983         14         w7gotchapc            UPLOAD       ERROR
986         14         w7gotchapc            UPLOAD       ERROR
997         14         w7gotchapc            UPLOAD       ERROR
1001        14         w7gotchapc            UPLOAD       ERROR
```

Check on a Command

```
$ lerc_ui.py -c 1002

----------------------------
Command ID: 1002
Client ID: 14
Hostname: w7gotchapc
Operation: RUN
  |-> Command: cd "C:\Program Files (x86)\Integral Defense\" && Del "C:\Program Files
↪(x86)\Integral Defense\w7gotchapc_dirOfInterest.zip"
  |-> Async: False
Status: COMPLETE
Client Filepath: None
Server Filepath: data/W7GOTCHAPC_RUN_1002
Analyst Filepath: None
File Position: 0
File Size: 84
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search