Industrial Control Systems Automated Digital Forensics Harvester

Final Report | Report Number 22-15 | January 2022

NEW YORK STATE | NYSERDA

NYSERDA's Promise to New Yorkers:

NYSERDA provides resources, expertise, and objective information so New Yorkers can make confident, informed energy decisions.

Our Vision:

New York is a global climate leader building a healthier future with thriving communities; homes and businesses powered by clean energy; and economic opportunities accessible to all New Yorkers.

Our Mission:

Advance clean energy innovation and investments to combat climate change, improving the health, resiliency, and prosperity of New Yorkers and delivering benefits equitably to all.

Industrial Control Systems Automated Digital Forensics Harvester

Final Report

Prepared for:

New York State Energy Research and Development Authority

Albany, NY

William Webb
Project Manager

Prepared by:

Electric Power Research Institute

Palo Alto, CA

Richard Alcalde
Piotr Lisowski
Zack Ankuda
Andrew Le
Eroshan Weerathunga
Nora Ajwad
Principal Investigators

NYSERDA Report 22-15 NYSERDA Contract 137944 January 2022

Notice

This report was prepared by The Electric Power Research Institute (EPRI) in the course of performing work contracted for and sponsored by the New York State Energy Research and Development Authority (hereafter "NYSERDA"). The opinions expressed in this report do not necessarily reflect those of NYSERDA or the State of New York, and reference to any specific product, service, process, or method does not constitute an implied or expressed recommendation or endorsement of it. Further, NYSERDA, the State of New York, and the contractor make no warranties or representations, expressed or implied, as to the fitness for particular purpose or merchantability of any product, apparatus, or service, or the usefulness, completeness, or accuracy of any processes, methods, or other information contained, described, disclosed, or referred to in this report. NYSERDA, the State of New York, and the contractor make no representation that the use of any product, apparatus, process, method, or other information will not infringe privately owned rights and will assume no liability for any loss, injury, or damage resulting from, or occurring in connection with, the use of information contained, described, disclosed, or referred to in this report.

NYSERDA makes every effort to provide accurate information about copyright owners and related matters in the reports we publish. Contractors are responsible for determining and satisfying copyright or other use restrictions regarding the content of reports that they write, in compliance with NYSERDA's policies and federal law. If you are the copyright owner and believe a NYSERDA report has not properly attributed your work to you or has used it without permission, please email print@nyserda.ny.gov

Information contained in this document, such as web page addresses, are current at the time of publication.

Preferred Citation

New York State Energy Research and Development Authority (NYSERDA). 2022. "Industrial Control Systems Automated Digital Forensics Harvester," NYSERDA Report Number 22-15 (added by NYSERDA). Prepared by Electric Power Research Institute (EPRI), Palo Alto, CA. nyserda.ny.gov/publications

Abstract

Digital forensics plays an important role in an electric utility's cyber security incident response process to help determine if a device could have been compromised, and if so, help identify how it was compromised and what the impact may have been. Most often, a forensics analysis of a device occurs during the postmortem analysis of an incident. The routine and automated collection of forensics evidence prior to an incident can potentially help identify issues with a device that could provide value earlier in the incident response process.

This report describes the development of an automated forensics harvesting tool designed to run on intelligent electronic devices (IEDs) in a utility's operational technology (OT) environment, such as substations. The tool extends the use of existing communication protocols and has adapted it for use in the acquisition of forensics artifacts. The wide variety of manufacturers and the embedded nature of devices used in a utility's OT environments can make it difficult for utility employees to perform forensics analyses of their own. The openness and flexibility of this harvester tool and protocol, if adopted, would facilitate the forensics acquisition processes and analysis across a wide variety of supported devices.

Keywords

cyber security, industrial control systems, digital forensics, incident management, forensics automation

Acknowledgments

The Electric Power Research Institute (EPRI) prepared this report.

Project Manager:
William Webb

EPRI would like to acknowledge the efforts of Consolidated Edison, The MITRE Corporation, and General Electric.

Principal Investigators:
Richard Alcalde (Consolidated Edison)
Piotr Lisowski (Consolidated Edison)
Zack Ankuda (MITRE)
Andrew Le (MITRE)
Eroshan Weerathunga (GE)
Nora Ajwad (GE)

EPRI would also like to acknowledge the support of the New York State Energy Research and Development Authority (NYSERDA).

Table of Contents

List of Figures

Acronyms and Abbreviations

ATT&CK	Adversarial Tactics, Techniques & Common Knowledge (MITRE)
BIOS	Basic Input Output System
EPRI	Electric Power Research Institute
FIFO	First In, First Out
GB	Gigabyte
ICS	Industrial Control System
IED	Intelligent Electronic Device
IETF	Internet Engineering Task Force
IT	Information Technology
LiME	Linux Memory Extractor
NETCONF	Network Configuration Protocol
NYSERDA	New York State Energy Research and Development Authority
OT	Operational Technology
RFC	Request for Comments
RPC	Remote Procedure Call
SCP	Secure Copy Protocol
SNMP	Simple Network Management Protocol
SSH	Secure Shell
UEFI	Unified Extensible Firmware Interface
URI	Uniform Resource Indicator
USB	Universal Serial Bus
WSL	Windows Subsystem for Linux
XML	Extensible Markup Language
XMLNS	XML Namespace

Executive Summary

The key question addressed in this report focuses on the development of a prototype automated digital forensics harvesting tool and assesses its potential benefits when operating in a utility's environment. Initial research defined the requirements for collecting forensics evidence of interest from a variety of devices and identified the forensics capabilities of the hardware device used for this project.

The GE G500 substation gateway, released in 2019, is a Linux-based platform that uses Docker containers to isolate the device's various applications. As industry trends point to a shift from dedicated embedded platforms to more conventional operating systems environments, this device enabled the development team to not only explore the benefits of automated forensics acquisitions in a Linux environment, but also one that used a containerized environment.

The prototype harvester securely communicates with a client using a derivation of the Network Configuration Protocol (NETCONF) over a secure shell (SSH) subsystem. The remote procedure call (RPC)-based protocol provides a structured messaging format using defined namespaces in which commands can be sent. The protocol's inherent flexibility provides extensibility that promotes adoption by device manufacturers and customization based on a device's capabilities.

By communicating with the harvester—both for sending commands and receiving data—using a standardized and flexible communications protocol, provides a foundation that enables the potential use of automation tools to routinely capture forensics information from a device and assess for changes that could indicate a potential issue.

1 Introduction

Over the past several years, the number of cyberattacks on industrial control system (ICS) devices has increased. Additionally, with the rapidly changing cyber security threat, the ability to collect critical forensics artifacts quickly and accurately from ICS devices within an organization's operational technology (OT) environment has become a crucial component in an organization's incident response process.

The analysis of such forensics data may help investigators identify the mechanisms by which one or more devices became compromised along with the attack vectors that were used to compromise these devices. Most forensics analyses historically have occurred as a postmortem of a suspect device in the later stages of the incident response process. Additionally, the embedded nature of the majority of ICS devices, which often function in custom operating systems, can make it difficult to perform these analyses, especially when there is no interface provided in which an investigator may forensically examine the device. In recent years many next-generation ICS devices have been running operating systems such as Linux, which allows forensics investigators to use tools and techniques from more established enterprise and information technology (IT) environments to aid in the analyses.

1.1 Goals

The objective of this research is to examine the feasibility of automated forensics collection and its potential support for collecting evidence while a device remains in an operational state. Additionally, a prototype forensics harvesting tool was developed to harvest evidence and house it in an organized manner to facilitate analysis by forensics investigators. A key component of this research was the development of a communications protocol that provides a unified structure for sending commands and receiving data.

1.2 Harvester Requirements

The forensics harvester tool is a program, primarily written in Python, that is designed to run on the target device, or any other Linux environment that supports the Python language. Other ancillary scripts and tools may be required and may be written in other languages such as bash script. For this research device, several bash scripts facilitate evidence collection, starting and stopping of various processes, and mounting and unmounting a Universal Serial Bus (USB) flash drive.

The overall harvester prototype operates using a client-server architecture, with the server running locally on the target device. The client will securely connect to the server to issue commands and receive data. Commands and the data returned follow a Remote Procedure Call (RPC) protocol developed for this effort. The client serves as the tool set for retrieving data and storing it in a database using a structure that makes it easily ingestible into other analysis tools.

1.3 Research Device

General Electric (GE) served as the industry partner for this project, and as such the project team selected the G500 Advanced Substation Gateway to use during the undertaking to create the forensics harvester prototype. The G500 is a device in GE's Multi-function Controller Platform, which provides a substation-hardened set of modular hardware and software components. The front and rear panels are shown in Figures 1 and 2, respectively. The device contains a 64-bit AMD x86-64 processor, 8 GB of RAM, a 128 GB solid state drive, multiple USB and network ports, and two video display ports.

Figure 1. GE G500 Substation Gateway Front Panel

Figure 2. GE G500 Substation Gateway Rear Panel

As is becoming more common in this device class, the G500 runs a customized Linux kernel. Additionally, the G500 sandboxes have different components and applications using OS-level virtualization, which is an operating system paradigm in which the kernel allows the existence of multiple isolated user-space instances, frequently referred to as containers. On Unix/Linux-based operating systems, this capability is based off an advanced implementation of the standard chroot mechanism, which changes the root folder for processes running in the container. Containers share the same kernel as the host operating system.

One of the early reasons for using containers was to better allocate resources, and to enable system administrators to consolidate multiple servers onto fewer servers that support containers. OS-level virtualization usually requires less system overhead than full-virtualization platforms. Although not ironclad, another benefit of using containers is improved security. The means by which containers may interact with each other or the host is through shared folders or volumes, or via a networking stack. This network stack can be managed by a software firewall, so administrators can restrict inter-container and container-host communications.

One of the most popular platforms that uses OS-level virtualization is Docker. Founded in 2013, Docker has become a widely used containerization platform in IT environments. Less common is its use in OT environments, and to the project team's knowledge, the G500 is the first device of its class to use Docker and OS-level virtualization. Given the device's unique operating environment and the expectation that more devices in the future will use containers, the G500 quickly became the project team's research device of choice.

1.4 Device Forensics Capabilities

The device's Linux operating system, along with its software, provides valuable information that would be useful in a forensics investigation. Running a conventional Linux operating system affords forensics investigators the use of many of the common forensics investigative tools that are commonly used in IT and enterprise environments. As such, the forensics harvester should implement a subset of these tools or extract the equivalent data that these tools usually retrieve.

The forensics capabilities of the research device were previously defined in the project's Task 2 deliverable. This includes using common Linux command line tools to capture information such as process lists, system trace calls and signals, process information, hardware information and diagnostics. Additionally, a wide variety of device information, such as configuration, connected assets and devices,

network information, firewall information, active and inactive network connections, running system services, a variety of system and Docker logs, and much more can be collected. This information is gathered into a folder on the device and is then compressed for extraction.

Disk images may be extracted using the '`dd`' command. While this will not preserve an exact snapshot of the disk, as files may still be changing as the disk is read, it should provide sufficient information to do an initial triage of the device. Memory is best extracted using the Linux Memory Extractor tool (LiME); this is accomplished by installing the LiME kernel module and capturing the image to file. This module is highly dependent on the operating system's kernel version and must be compiled to match. This kernel module shouldn't run continuously unless a memory image is requested, as not to unnecessarily consume system resources. Instead, LiME should only be loaded when the harvester is commanded to capture memory.

1.5 Extraction Tool (Client) Requirements

The extraction tool, or client, developed for this project must provide data in a format that can easily be ingested into analysis tools. This is important because it allows these tools to quickly read the captured data and begin making assessments as to why the device may be mis-operating.

The client does not manipulate the results from commands run on the target device. For example, the process list command has a consistent data structure that is common among most Linux environments. The harvester could manipulate this data structure and further break it down into a different, perhaps more easily readable, format for ingestion into analysis tools. However, it likely does not make sense to do so, as the data structure for the process list is already well understood and is trivial for any analysis tool to import the data in its original structure. This goes for nearly every standard Linux command that the harvester would execute. As such, it is sufficient for the harvester to store the output of commands, such as process lists and system diagnostics, and make this output readily available for analysis tools.

File attributes are a type of metadata that describes information about files and directories in a device's file system. Common file attributes include parameters such as file size, read/write/execute permissions, the dates that the file was create and last modified, and whether a file is visible. This information is very important to investigators when examining a device's file system. Attributes for a file should be preserved when extracted, whether via a total disk image, or if individual files are retrieved through the harvester.

Files and other information that are collected by the harvester should include some sort of integrity indicator to ensure that the file has not been modified after collection and that the file was correctly transferred across the network connection. Hashes are one possible method of validating file integrity. The goal of a hashing algorithm is to calculate a unique block of data that will result in a significantly different value when small changes are made to the input. This is helpful not only to verify the integrity of the file, but to ensure that not even a single bit of data was changed in the original file during transmission. Hashes are popular because they use one-way hashing algorithms; in other words, it is easy to compute a hash of a file, but virtually computationally impossible to recreate the file based on its hash. One of the more popular hashing functions is MD5. Developed in 1992, weaknesses in the hashing function make it suitable now only to verify data integrity. More modern hashing functions such as the SHA-2 family, developed by the National Security Agency, may be more appropriate for use in the harvester. It is important to note that hashes are not designed to provide data authenticity, which is accomplished with the use of certificates.

2 Harvester Development and Architecture

The harvester suite consists of both a server that runs on the device and a client that runs on any Linux-based computer that has Python 3.8 or higher installed; if the analyst intends to run the client on a Windows-based machine, then the Windows Subsystem for Linux may be used. Given to the device's containerized architecture, the project team felt it appropriate to package most of the harvester's core capabilities within a container. Additionally, because of the inherent isolation provided with containerization, certain files must reside on the host operating system so that the harvester can otherwise access host operating system files, access other containers, and execute commands from the host.

2.1 Harvester Server

The harvester server exists on the device in two components: (1) a Docker container and (2) supporting files on the host operating system.

2.1.1 Docker Container

The harvester's primary functionality exists in a Docker container, which runs on the Ubuntu 16.04 platform (Xenial Xerus) and uses Python 3.5. This container provides Secure Shell (SSH) access into a special subsystem that launches the harvester server's main Python code.

The main Python script is responsible for communicating both with the client and the processes running on the host operating system. The `Dockerfile` script used to assemble this container is provided in appendix A.

2.1.1.1 Secure Access and Authentication Keys

Communications between the client and the server occur over an encrypted SSH connection. Most often, SSH users login to a remote system using a username and password. For added security, SSH security keys can be employed in lieu of a password (often referred to as password-less logins). As such, the harvester server is configured only to utilize security keys. The client's public key is built into the harvester container and allows the client to use its private key to connect to the harvester server subsystem. The harvester's container image must be rebuilt and loaded onto the device if the user's access keys change.

2.1.1.2 Host File System Access

When the harvester's container is launched from the host operating system, two mount points are created that allows the client to access the host's file system:

- Host mount point: `/` (read only access, appears in harvester container as `/tmp/host_root_fs`)
- Host mount point: `/tmp` (read/write access, appears in harvester container as `/tmp/host_tmp`)

While the harvester provides the ability to transfer files from the host directly to the client via its communications protocol, there is a 33–36% overhead on the file size transferred due to the encoding to Base64 before the file is transferred. This overhead can often be irrelevant for small files, especially when the client is connected to the device on a fast network connection. For large files, users may prefer to connect to the harvester's container using the secure copy protocol (SCP), which uses the same SSH security keys as the client. This protocol supports binary transfers and can be significantly faster to transfer large multi-megabyte and gigabyte files. Additionally, there are many graphical clients, such as WinSCP (on Windows), that provide a graphical interface in which the user can browse the host's file system and retrieve files of interest.

The host system provides read/write access to its `/tmp` folder via the harvester container. This is to allow the harvester to send authorized commands to the host and to allow the host to communicate with and send data to the harvester via named pipes; this is done in lieu of a network-based communication method. Details on container-host communications are discussed in Section 2.1.5.

2.1.2 Host Files

In addition to the harvester container image that resides on the host, the host also contains several files to facilitate collection of forensics information. These files are listed in Table 1.

Table 1. List and Purpose of Harvester Files on the Host

File	Purpose
`/home/harvestor/rpcServer-host.py`	Host harvester Python script.
`/home/harvestor/config.ini`	Command whitelist (see Section 2.1.4).
`/home/harvestor/conSSH.sh`	Starts harvester container and host Python script.
`/home/harvestor/lime-standard.ko`	LiME memory extraction kernel module.
`/home/harvestor/mount_harv_ext_st.sh`	Script that mounts USB flash drive.
`/home/harvestor/umount_harv_ext_st.sh`	Script that unmounts USB flash drive.
`/home/harvestor/all.sh`	Script to execute all whitelisted commands, and stores output to `/tmp` or USB flash drive.
`/home/harvestor/copyetc.sh`	Script to copy /etc folders from host and all containers, stores output to `/tmp` or USB flash drive.
`/home/harvestor/filedetails.sh`	Script to list detailed file information and statistics, stores output to `/tmp` or USB flash drive.
`/home/harvestor/run_lime.sh`	Script to insert the LiME memory module into the host's operating system kernel and dumps the RAM to USB flash drive.

2.1.3 Command Whitelist

For security purposes and because it breaks the inherent security and isolation provided by the use of Docker containers, it is not ideal to allow users to execute arbitrary commands directly on the host via the harvester's Docker container. As such, the project team developed a whitelist of authorized commands that the harvester client can execute. The list, contained in a file called `config.ini`, contains 240 commands that collect a large amount of information both from the host operating system and the other Docker containers running on the device across a variety of categories including:

- Docker engine information (host only; 22 commands)
- AppArmor security module status (host only; 1 command)
- Kernel parameters (host and Docker containers; 6 commands)
- Log files (host and Docker containers; 19 commands)
- Network statistics and information (host and Docker containers; 23 commands)
- Process information (host and Docker containers; 33 commands)
- System information (host and Docker containers; 89 commands)
- User information (host and Docker containers; 42 commands)
- Ancillary shell scripts (such as RAM capture and detailed file information; 5 commands)

This command list is only editable when accessing the device's host environment. From the perspective of the harvester container, this file is read-only, and thus prevents arbitrary command execution if the harvester client's SSH keys were compromised.

2.1.4 Container-Host Communications

Bi-directional container-to-host communications are accomplished using a concept called named pipes. Also known as a FIFO (First In, First Out), named pipes are one of the most common methods of inter-process communications on the same machine. Named pipes provide a method to send and receive data between processes without having the performance penalty of involving the network stack.

By providing a read/write mount point to `/tmp` on the host system (and to `/tmp/host_tmp` on the harvester container), two named pipes are created for the harvester container to send commands to the host, and for the host to send data to the harvester container. These named pipes are called name `/tmp/cetx` and `/tmp/cerx`. While they appear as normal files in the file system, they contain a special pipe flag (shown in directory listing below in Figure 3, the pipe flag '`p`' highlighted in red) in the file's permissions to indicate that they are a named pipe.

Figure 3. Named Pipes for Harvester Communications

```
root@linux:/tmp# ls -al
drwxrwxrwt    7 root     root           220 Oct 10 19:28 .
drwxr-xr-x   20 root     root          4096 Oct  5 12:25 ..
drwxrwxrwt    2 root     root            40 Oct  8 01:26 .ICE-unix
drwxrwxrwt    2 root     root            40 Oct  8 01:26 .Test-unix
-r--r--r--    1 root     root            11 Oct  8 01:27 .X0-lock
drwxrwxrwt    2 root     root            60 Oct  8 01:27 .X11-unix
drwxrwxrwt    2 root     root            40 Oct  8 01:26 .XIM-unix
drwxrwxrwt    2 root     root            40 Oct  8 01:26 .font-unix
-rw-------    1 root     root             0 Oct 10 10:37 .python-history
prw-------    1 root     root             0 Oct 10 12:07 cerx
prw-------    1 root     root             0 Oct 10 13:17 cetx
```

The two named pipes are created when the harvester is launched and persist until the system is rebooted.

2.2 Harvester Client

The harvester client is the command line tool that interfaces with the harvester server. It is responsible for sending commands to the server and receiving and storing responses. The client is written in Python and supports versions 3.8 and higher. Some of the packages used to develop the client require a Linux operating environment; this could also include a Linux-based virtual machine or using Windows Subsystem for Linux (WSL). Native MacOS compatibility was not evaluated.

All connections, commands, and responses are logged in a relational database to provide a standardized structure. This database can then be queried by analysis tools. For example, the query could return all versions of a configuration file that was periodically downloaded over time and be analyzed for unauthorized or potentially harmful changes that could warrant further investigation.

The client and its packages run in a virtual environment using `pipenv`. This makes it easy to install the required packages automatically on new machines and creates a deterministic environment in which the client will reliably operate. A Markdown-formatted `README` file provides instructions on how to prepare the client for connecting and communicating with the target device.

2.2.1 Interface

The client operates in a command line environment, launched in a virtual Python environment from the operating system's terminal. A screenshot of the harvester client's interface is shown in Figure 4. Colors are used to enhance the client's visual appearance and to highlight the components of the various commands and responses sent to and received from the harvester server. Therefore, use of a terminal environment that supports a minimum of 256 colors is recommended.

Figure 4. Harvester Client Interface

2.2.2 Configuration File

A configuration file called `config.toml` allows the user to specify some basic client operating parameters. This includes items like the type and location of the client's database, various logging options, and whether the user wishes to enable the interactive menu system or execute commands manually.

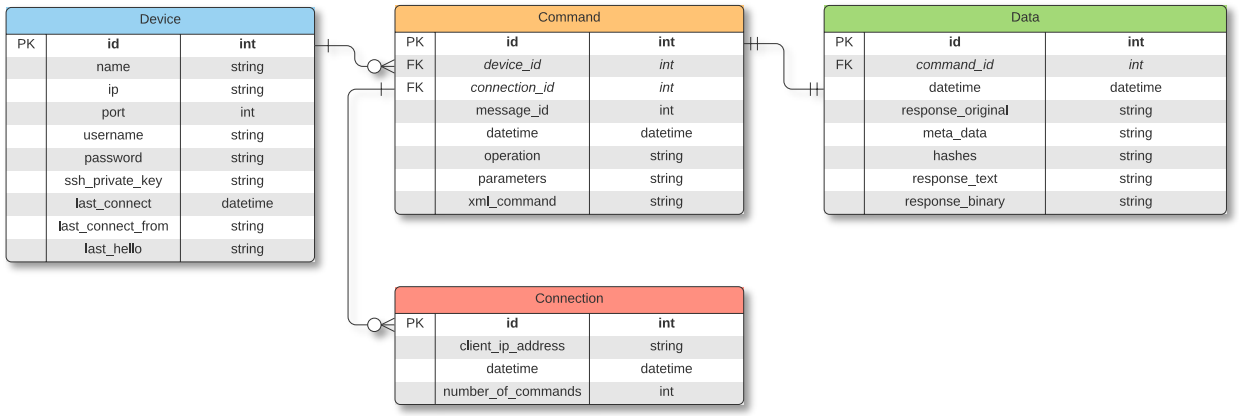
2.2.3 Database

The client's database serves as the primary data logging mechanism and is responsible for storing a variety of information, including the following:

- **Device information**—a list of all devices the client can connect to, including its IP address, login credentials (username, password, SSH private key), the last connection date and time, and the harvester server's most recent `HELLO` message.
- **Connection log**—a log of all connections made by the client and how many commands were executed during the session.
- **Command log**—a log of each command sent to the harvester server, including the date and time, the operation, command parameters, and the XML-formatted message string sent via the communications protocol.
- **Data log**—a log of all responses received from the harvester server, including the date and time, the XML-formatted message string received via the communications protocol, the decoded output from the data payload, and optional metadata and hashes included in some responses.

Additional fields, specifically unique IDs, are associated with each data point for all of the parameters above to facilitate the relational nature of the database. Figure 5 shows the entity relationship diagram of the database's four tables, table columns, and the relationships between the tables.

Figure 5. Harvester Client Database's Entity Relationship Diagram



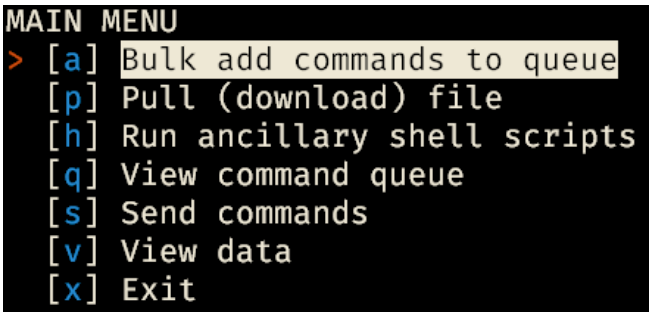
The client currently supports the SQLite relational database. SQLite is a serverless platform and is lightweight, fast, and all data is stored in a single database file. Despite the client only supporting SQLite, the database abstraction module used in the client's code (SQLAlchemy) would make transitioning to a different relational database platform (such as Postgres, MySQL/mariadb, or MS SQL Server) a trivial exercise.

There are many command-line and graphical tools that support querying and viewing SQLite databases. Additionally, as demonstrated with the client itself, many programming languages provide bindings to SQLite, making accessing data in the database easy for any custom analysis tools. Another advantage of using a single-file relational database is portability. For larger, automated harvester client deployments, a separate server-based relational database management system would most likely be recommended.

2.2.4 Menu and Manual Commands

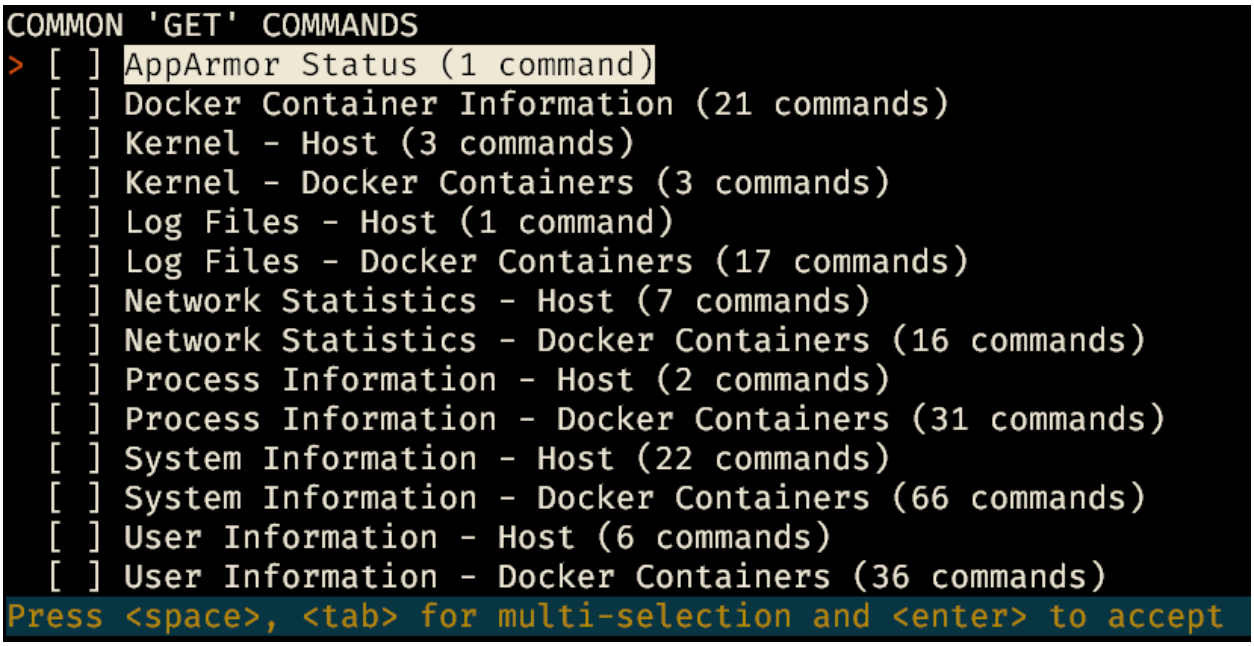
The client supports both an interactive menu system and the ability to programmatically send specific commands to the harvester. Switching between the two operational methods is achieved by enabling or disabling the `enable_menu` setting in the client's configuration file. The main menu is shown in Figure 6.

Figure 6. Harvester Client Main Menu



As mentioned in Section 2.1.4, the device's harvester server supports 240 various commands. To facilitate the ability to view and execute these commands, the menu system categorizes the various commands in 14 different command groups. The user can arbitrarily select which command groups they wish to execute. The various command groups are shown in Figure 7.

Figure 7. Device Harvester Command Groups

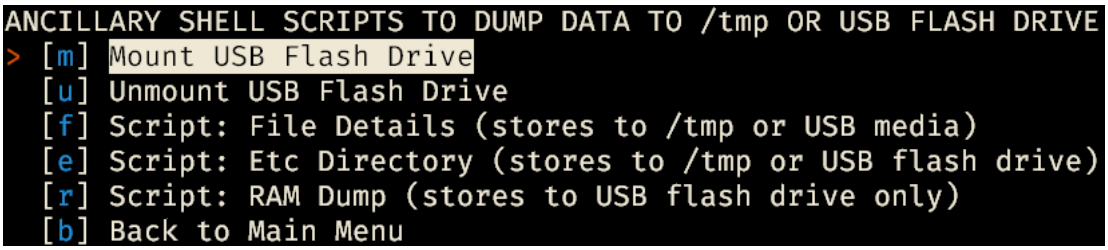


The harvester server can also retrieve files from the device through the harvester protocol. To demonstrate this capability, a menu item allows the user to enter an arbitrary file path and file name to retrieve the file (as an example, `/etc/passwd`).

Commands selected from the command group and individual pull file commands are added to a queue. This queue can be optionally viewed to show each command's XML-based message that will be sent to the harvester server. Once the user has added all the desired commands to the queue, the commands can then be sent to the harvester server. At this point, the responses are received and stored in the client's database.

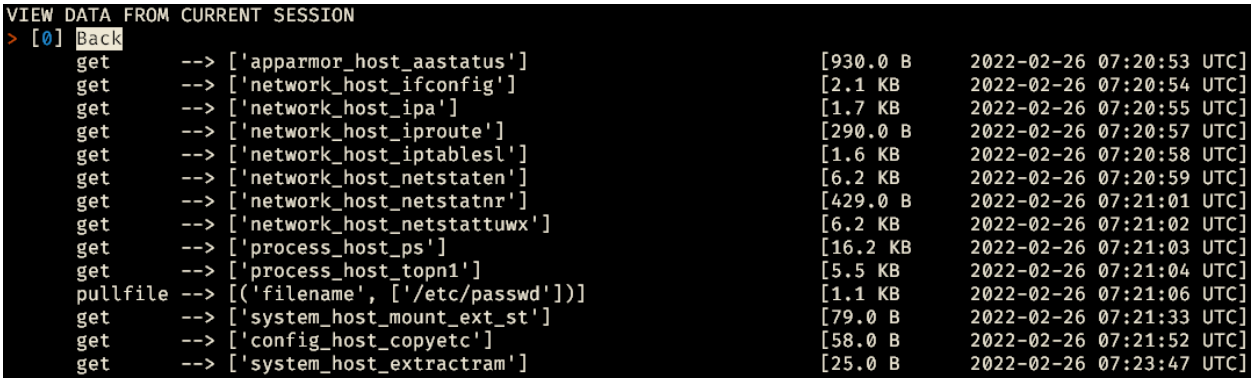
The harvester server, in addition to the commands included in the command groups, supports various ancillary shell scripts that capture additional data and perform various actions. These shell scripts are available in the client (Figure 8); unlike the command groups, these shell scripts are not added to the command queue and are executed in real time.

Figure 8. Ancillary Shell Script Menu



The client also contains a basic data viewer to display responses from the harvester server (Figure 9). The client data viewer only shows data from commands executed within the current session. Selecting a command will show the decoded response and any attributes included in the response, such as file attributes and file hashes for pull file commands. Despite only showing data from the current session, data from all sessions is still available in the database file.

Figure 9. Harvester Client Data Viewer



3 Communications Protocol

The Harvester protocol provides a method for embedded devices to allow the extraction of system artifacts to aid in incident response and digital forensics. The protocol is based on NETCONF, a common network management protocol. This section discusses the background and decision process for selecting this protocol from which to develop the Harvester protocol.

3.1 NETCONF Protocol Background

NETCONF is a network management protocol developed by the Internet Engineering Task Force (IETF) in 2006, and later revised in 2011. NETCONF was created to address several shortcomings of the 1980s Simple Network Management Protocol (SNMP). Despite its intended use, SNMP was and still is used as a network monitoring protocol, but very few manufacturers used it to configure their devices.

Instead of using SNMP, historically, network and other telecom equipment providers used proprietary command line interfaces to configure their products. This was often preferred since it was text based, as opposed to following the basic encoding rule structures of SNMP. In fact, few manufacturers provided the capability to configure their products using SNMP. As a result, the IETF worked with manufacturers, such as Juniper, who had already been using a network management approach that used XML to create a Request for Comments (RFC) that later became the approved NETCONF protocol.

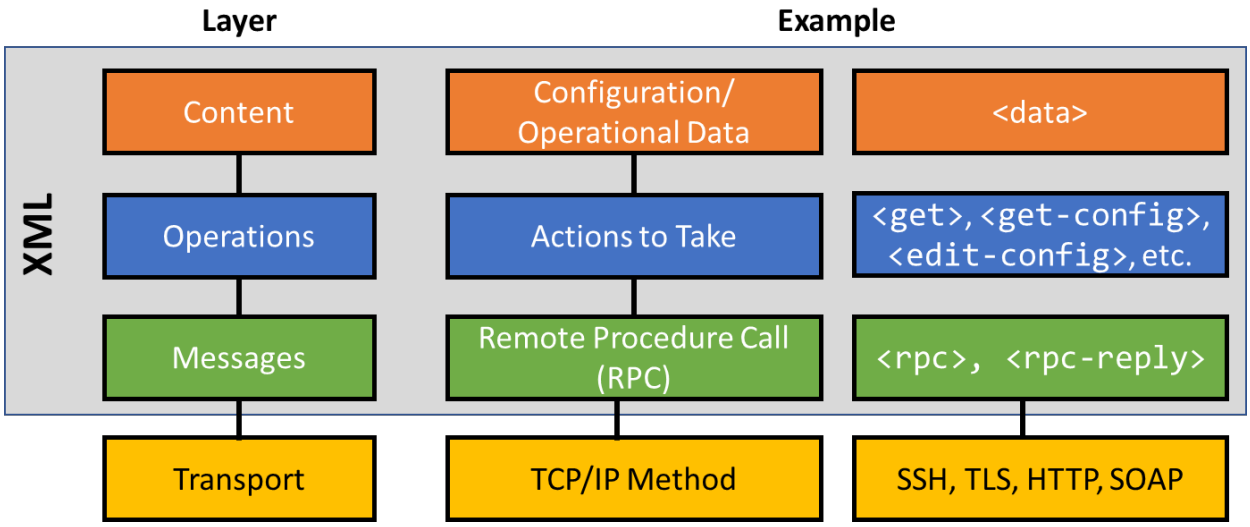
3.2 Why NETCONF?

NETCONF operations are well-formed XML messages. XML provides an inherent structure to messages that are both sent and received, which is useful for receiving artifacts from the target device. NETCONF's architecture allows it to be expanded to add additional capabilities. Additional protocol features (or capabilities) that a device supports are communicated between the server and the client during a capability exchange (which occurs during the `HELLO` message). This extensible capability will be useful for the harvester, as device manufacturers can choose to add or remove capabilities from their implementation of the harvester based on the capabilities and limitations with their products. For example, not every device may be capable of performing memory or disk image dumps, and as such this capability would not be advertised.

The NETCONF protocol is partitioned into four layers, shown in Figure 10:

- **Content Layer:** consist of configuration and operational data (such as notifications).
- **Operations Layer:** provides base protocol operations to retrieve data.
- **Messages Layer:** the mechanism for encoding remote procedure calls (RPC).
- **Transport Layer:** provides a secure and reliable transport vehicle for messages sent between the client and the server.

Figure 10. NETCONF Protocol Layers



3.3 NETCONF Structure and Communications

NETCONF messages provide a simple XML framing mechanism for coding commands and responses:

- RPC commands: `<rpc>`
- RPC replies: `<rpc-reply>`
- Event notifications: `<notification>`

The harvester extends the NETCONF protocol by implementing a variety of commands for the specific use of gathering forensics information. The harvester protocol is designed to be flexible to allow other manufacturers who adopt this protocol to add their own capabilities or disallow pre-defined commands.

NETCONF messages are encoded in XML using the UTF-8-character set. For SSH, a special 6-character message termination sequence is used to provide message framing: `]]>]]>`

3.4 Remote Procedure Call

NETCONF uses a simple RPC-based mechanism to facilitate communication between the server and the client. Requests to the server must be wrapped in an `<rpc>` tag, which includes a unique `message-id` attribute with an ASCII number as a field representing a specific message number. The server will reply with a response wrapped in an `<rpc-reply>` tag, which uses the same `message-id` attribute to indicate the command to which it is responding. RPC messages should also contain the XML Namespace (XMLNS). Inside the RPC tag are sub elements that contain the actual requests or functions that are sent to the server.

3.5 Harvester Protocol Commands

3.5.1 <hello>

Upon connecting to the harvester server, the server sends a `<hello>` message to the client that lists its capabilities. Capabilities are defined using namespace attributes as a uniform resource identifier (URI).

For example, Figure 11 shows the harvester server sends the following `HELLO` message.

Figure 11. Example of `HELLO` Message from Harvester Server

```
<hello xmlns="urn:EPRI:params:xml:ns:harvestor:base:1.0">
  <capabilities>
    <capability>urn:EPRI:params:xml:ns:harvestor:base:1.0</capability>
    <capability>urn:EPRI:params:xml:ns:harvestor:capability:device:1.0</capability>
    <capability>urn:EPRI:params:xml:ns:harvestor:capability:pullfile:1.0</capability>
  </capabilities>
</hello>
```

The root message contains the harvester's base namespace and two capabilities further identified by their own namespaces: `device` and `pullfile`. This effectively lets the client know what commands are available for use.

3.5.2 <pullfile>

This command requests the server send a file over the active connection to the client. The request should include a `<filename>` tag whose data contains a base64-encoded string of the full path to the file. The server will respond with a similar `<filename>` tag with the same base64-encoded filename path. File attributes such as the file size, modification date, and permissions are included in a `<metadata>` tag. Various hashes, such as MD5, SHA1, and SHA256 are included in a `<hash_values>` tag.

An example command to retrieve `/etc/passwd` in Figure 12. Note that the file path and file name have been encoded in base64.

Figure 12. Example of a Formatted PULLFILE Command

```
<rpc xmlns="urn:EPRI:params:xml:ns:harvestor:base:1.0" message-id="101">
  <pullfile xmlns="urn:EPRI:params:xml:ns:harvestor:capability:pullfile:1.0">
    <filename>L2V0Yy9wYXNzd2Q=</filename>
  </pullfile>
</rpc>
```

A simplified response is shown in Figure 13, with a truncated metadata, hash values list, and a truncated data payload.

Figure 13. Example PULLFILE Response with File Contents and Attributes

```
<rpc-reply message-id="101" xmlns="urn:EPRI:params:xml:ns:harvestor:base:1.0">
  <pullfile xmlns="urn:EPRI:params:xml:ns:harvestor:capability:pullfile:1.0">
    <filename>L2V0Yy9wYXNzd2Q=</filename>
    <metadata access_time="1643519746" change_time="1590568660" />
    <hash_values sha1="3f17a0eb49e4b6581f2757565fe2cfac1c0400c8" />
    <base64-data>cm9vdDp4OjA6MDpyb290Oi9ob25leGlzdGVudDovYmluL3NoCg==</base64-data>
  </pullfile>
</rpc-reply>
```

3.5.3 Device-Specific Capabilities

The extensible harvester protocol enables manufacturers to include their own capabilities. A device-specific capability is shown in the `HELLO` message with the namespace `urn:EPRI:params:xml:ns:harvestor:capability:device:1.0`. Commands using this namespace correspond to the 240 whitelisted commands discussed in Section 2.1.4.

For example, Figure 14 shows a `<get>` command using this namespace is requesting the server provide a response to the `apparmor_host_aastatus` command, which provides information about the AppArmor kernel security module running on the host operating system.

Figure 14. Example of a Formatted GET Command Using Device-specific Capability

```
</rpc-reply>
<rpc xmlns="urn:EPRI:params:xml:ns:harvestor:base:1.0" message-id="102">
  <get xmlns="urn:EPRI:params:xml:ns:harvestor:capability:device:1.0">
    <apparmor_host_aastatus/>
  </get>
</rpc>
```

The response, shown in Figure 15, includes the command tag (<apparmor_host_aastatus>) which contains a <base64-data> tag to indicate that the response has been encoded in base64.

Figure 15. Example GET Response

```
<rpc-reply message-id="102" xmlns="urn:EPRI:params:xml:ns:harvestor:base:1.0">
  <get xmlns="urn:EPRI:params:xml:ns:harvestor:capability:device:1.0">
    <apparmor_host_aastatus>
      <base64-data>YXBwYXJtb3IgbW9kdWxlIGlzIGxvYWRlZC4K5lZC4K</base64-data>
    </apparmor_host_aastatus>
  </get>
</rpc-reply>
```

4 Testing, Validation, and Results

The General Electric (GE) Multilin G500 Substation Gateway is a next generation Industrial Control System and network security appliance designed to securely connect substation computer components and the larger power systems network. Electrical substations contain important equipment to manipulate and distribute electrical power as it is transported and delivered to end customers. Protecting this equipment is important to the health of the electrical grid, its components, and the operating company's financial stability. In recent years electrical power systems have been targeted by cyber attackers because of societal dependency on their operation. The cyber attacks in Ukraine in 2016 and 2017 that caused mass power outages in the capital city, Kiev, illustrates how far cyber attackers are willing to go to disrupt, degrade, and destroy electrical power capabilities.

Protecting substation components from cyber attackers is important and providing a security appliance is a great first step, but additional steps are required to adequately protect the security appliance. With a security appliance in front of substation components, the security appliance is a new target for attackers. Therefore, the security appliance must also be monitored for intrusion and malicious activities. To help mitigate the potential effects of intrusion, the Electric Power Research Institute (EPRI) partnered with Consolidated Edison, MITRE, and GE to develop an automated forensics collection tool for a device. The forensics tool, known as the Forensics Harvester, is a custom collection software designed to collect forensics evidence from the device through a USB Drive or network protocol.

The Forensics Harvester runs autonomously on the device and collects evidence from several sources including the command line, Random Access Memory (RAM), and the file system. MITRE developed 65 test cases involving critical evidentiary locations within the device to verify the effectiveness of the Forensics Harvester. The test cases are based on MITRE's ATT&CK frameworks and use locations within the device that will most likely be targeted by adversaries. By selecting high-risk locations within the device for testing, MITRE shows that the Forensics Harvester can collect the data necessary to perform an effective forensics investigation.

The Forensics Harvester successfully passed 57 tests and partially passed three. There were five tests that the Forensics Harvester did not pass. The Forensics Harvester excelled at retrieving information directly from the host operating system itself and had several useful methods to extract information directly from the device. Through its evidence collection tests, MITRE also verified the Forensics Harvester's functionality, that is, namely remote collection, USB collection, access to a read only

version of the host file system, volatile Random Access Memory (RAM) collection, and output of interrogative Linux utilities required for a thorough forensics investigation. The Forensics Harvester software was robust and easy to install and operate.

In addition to the test results, MITRE identified additional capabilities and associated tests that would improve the Forensics Harvester's collection capabilities and aid in forensics analysis. Specifically, the collection of the following forensics artifacts would be useful to forensics analysis: a volatile memory collection, complete with a system map and a Unified Extensible Firmware Interface (UEFI) Basic Input/Output System (BIOS) with variable collection capability.

5 Conclusion

This research effort developed a unique forensics harvester tool that allows utility employees to extract important forensics information from a device in real time. The harvester communications protocol is an extension of an existing and supported communications protocol and is demonstrated to be an effective method of interacting with a device. The harvester, in addition to its protocol and the method by which data received is stored in a logical and structured format, accomplishes the project's primary goals of supporting forensics automation. The harvester prototype is well-suited (1) to be integrated in security orchestration and automation (SOAR) tools, (2) to continuously monitor devices for potentially malicious activities (3) to support automated threat detection and, perhaps in the future, (4) for supporting automated threat mitigation.

5.1 Lessons Learned and Best Practices

The project team consisted primarily of engineers with backgrounds in electrical engineering, cyber security, and forensics, and had limited experience with product development. The project's research and development (R&D) phase most certainly benefited more with the team's core background than it would have with a team of product development managers. Nevertheless, the project team learned some valuable insights on product development that could be valuable to similar projects in the future, as well as to any organizations that might intend to incorporate forensics capabilities into their products:

- **Security:** With the potential for cyber security attacks on critical infrastructure increasing every year, security remained a top priority for this project. The harvester prototype uses strong cryptographic authentication for access, and while the overall tool can obtain read-only access to sensitive (or restricted) parts of a device, the containerized architecture and command whitelist intentionally limits unauthorized access to the host operating system.
- **Prioritization of Features:** The project team excelled at identifying capabilities that would be useful in a forensics harvesting tool. However, the team learned that it was important to first build a capable, yet flexible, core tool that would operate on a multitude of environments. Then, as time allowed, the extensibility of the communications protocol and harvester core could make it easier to implement additional functionality and capabilities.
- **Continuous Learning:** As a research project to develop a prototype tool to fill a gap in ICS forensics gathering capabilities, the project team understood the importance of spending a significant amount of time exploring and learning about the best mechanisms by which the tool should be implemented.
- **Constant Communication:** The prototype's development roadmap divided and assigned various components to individual team members. The results of each team members' research could potentially impact other team members, so the entire team met frequently to discuss research and implementation strategies, holding additional breakout sessions as needed.

- **Embrace Debate, Diversity, and Discomfort:** The project team was able to create a feature rich, working prototype forensics harvester tool that will undoubtedly provide value to future forensics research and future forensics offerings in devices. The diverse background of team members led to valuable discussions and debate as features and implementation strategies were developed. Differing views and opinions must be appreciated and taken into consideration when working through the product development life cycle.

5.2 Technology Readiness Level

Prior to this research, efficient and effective methods of extracting forensics artifacts from an in-situ and operating device were not widely available. Through this project, an effective method was developed and demonstrated using both the harvester server and harvester client. The use of a database to organize commands and responses will facilitate the analysis of data. Manufacturers can consider the potential adoption of the harvester, develop a set of capabilities for their devices, and allow electric utility forensic analysts to interact with their devices using an extensible protocol.

5.3 Future Work

Additional capabilities that were discussed, but not implemented in the prototype could further expand the harvester's abilities to collect additional forensics evidence. Where a full-memory dump was demonstrated with the prototype harvester, the capability of extracting a full-disk image would provide immense value to analysts. Large files, such as with the memory dump, are not well suited for transferring over the harvester protocol due to their large size and the overhead associated with the required encoding for the protocol. While the prototype demonstrates how these large files can be stored to a USB flash drive using the harvester and its commands, additional data extraction methods to stream the data over an additional and more efficient communications channel would be beneficial. An example would be using `netcat` senders and listeners to stream the data to the client as it is extracted from the server.

Appendix A. Harvester Container `Dockerfile`

```
# Using Ubuntu 16.04, xenial-20210804, Python3.5
FROM ubuntu:xenial

# sshd_config_pub_on configures ssh daemon in the docker container and allows only
# for pub_key auth
# id_rsa_harvestor.pub is a public key for the client to be authenticated, it can be
# changed upon each build time
ARG mySshdConfig='./sshd_config_pub_on'
ARG myAuthorized_keys='./id_rsa_harvestor.pub'

# Set environment variable for non-interactive installation
ENV DEBIAN_FRONTEND="noninteractive"

# Install forensic tools, SSH daemon, text editor, create directory for storage,
# and clean up after installation
RUN apt-get update -y -q \
      && apt-get install forensics-all -y -q \
      && apt-get install openssh-server -y -q \
      && apt-get install vim -y -q \
      && mkdir /home/harvestor \
      && apt-get clean \
      && rm -rf /var/lib/apt/lists/*

# Change working directory
WORKDIR /home

# Place code in /home/harvestor/ directory
COPY ./harvestor_srv/* ./harvestor/

# The internal port number for SSH service, upon running the container -p 1022:22
# should be set to forward the port
EXPOSE 22

WORKDIR /home/harvestor

# Configure sshd to disable password and enable ssh keys, copy keys to user folder
# and set appropriate permissions
RUN cat $mySshdConfig > /etc/ssh/sshd_config \
      && mkdir /root/.ssh \
      && chmod 700 /root/.ssh \
      && cp $myAuthorized_keys /root/.ssh/authorized_keys \
      && chmod 600 /root/.ssh/authorized_keys \
      && chmod +x /home/harvestor/rpcServer-API.py

# Restart the SSH service and launch persistent shell to keep the container running
CMD service ssh restart && /bin/bash
```

NYSERDA, a public benefit corporation, offers objective information and analysis, innovative programs, technical expertise, and support to help New Yorkers increase energy efficiency, save money, use renewable energy, and reduce reliance on fossil fuels. NYSERDA professionals work to protect the environment and create clean-energy jobs. NYSERDA has been developing partnerships to advance innovative energy solutions in New York State since 1975.

To learn more about NYSERDA's programs and funding opportunities, visit nyserda.ny.gov or follow us on Twitter, Facebook, YouTube, or Instagram.

New York State Energy Research and Development Authority

17 Columbia Circle
Albany, NY 12203-6399

toll free: 866-NYSERDA
local: 518-862-1090
fax: 518-862-1091

info@nyserda.ny.gov
nyserda.ny.gov

State of New York
Kathy Hochul, Governor

New York State Energy Research and Development Authority
Richard L. Kauffman, Chair | Doreen M. Harris, President and CEO