A COMMON AUTOMATION FRAMEWORK FOR CYBER-PHYSICAL POWER SYSTEM STUDIES

An Undergraduate Research Scholars Thesis

by

ETHAN COPE

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Katherine Davis

May 2023

Major: Electrical Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Ethan Cope, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my faculty research advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

Page

ABSTRACT

A Common Automation Framework for Cyber-Physical Power System Studies

Ethan Cope
Department of Electrical and Computer Engineering
Texas A&M University

Faculty Research Advisor: Dr. Katherine Davis
Department of Electrical and Computer Engineering
Texas A&M University

As the power grid becomes more complex and integrated with communication networks, cyber-attacks become an increasingly relevant threat. CYPRES (Cyber Physical Resilient Energy Systems) is a system for simulating cyber and physical attacks on a power grid by comprehensively modeling a cyber-physical hardware-in-the-loop power system. However, the CYPRES system is tedious to spin up, as multiple sub-systems of the framework involve interdependencies across different applications. This paper details a common automation framework for the CYPRES system that both removes this manual overhead from running the system and drastically improves CYPRES' initialization speed. With the Jenkins, a continuous integration/continuous development tool, the author has established this automation infrastructure, *RESAuto*, for CYPRES system and demonstrated the benefits and effectiveness of automating manually intensive sub-systems operations with much simpler operations and less time consumption. This automation system makes it much easier to test more complex threat scenarios for larger, more realistic, and manually intensive scenarios. In addition to this, RESAuto also provides a real-time dashboard that shows simulation status and network traffic, allowing users to monitor RESLab's distributed computing resources at a glance.

DEDICATION

This paper is dedicated to my parents, my teachers, God, and the CPMA team, who made working on this project feel effortless.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Davis, and Hao Huang and Abhijeet Sahu, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues in the CPMA team for making my time at Texas A&M University a great experience.

Special thanks also to the open source community, specifically those behind Jenkins, expect scripting, and AutoHotkey: without such tools, this project would simply not exist.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

The authors would like to acknowledge the US Department of Energy Cybersecurity for Energy Delivery Systems program under award DE-OE0000895, the National Science Foundation under Grant 1916142, and the Texas A&M Undergraduate Research Scholarship for their support.

NOMENCLATURE

API	Application Programming Interface
ARP	Address Resolution Protocol
CI/CD	Continuous Integration / Continuous Development
CORE	Common Open Research Emulator
CYRPRES	Cyber Physical Resilient Energy Systems
DDOS	Distributed Denial of Service
DNP3	Distributed Network Protocol 3
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
LTS	Long Term Service
MITM	Man in the Middle
RTAC	Real Time Automation Controller
SSH	Secure Shell
TCP/IP	Transmission Content Protocol / Internet Protocol
VM	Virtual Machine

1. INTRODUCTION

The field of power grid simulation is by no means a new one. One of the most common power system simulators, PowerWorld, was first released in 1994. Simulations of this kind are commonly used to model the physical interconnections of the power grid accurately. However, the so-called “Smart Grids" of today are networks of interconnected systems that can include everything from load shifting to pricing to user-interaction with home area networks, etc.; such properties make modern power systems a far cry from the more isolated systems of past grid iterations. Colak has defined smart grids as self-sufficient systems that allow integration of any type and any scale of generation sources to the grid and that reduce the workforce, targeting sustainable, reliable, safe and quality electricity to consumers [1]. The benefits of smart grids are many, but the increasing linkage of the power grid with communication networks also lends itself to new forms of cyber-attack. John *et al.* posit that the power system is made even more vulnerable to cyber-attacks due to the increasingly open architecture of electric power communication networks and the utilization of unsecured standard IP based protocols [2]. Threats of concern include that an adversary can re-distribute power away from loads that need it, de-couple the network linking the grid together, or even shut down generation facilities without needing physical access to these targets themselves. A modern power simulation must also model these cyber integrations if it hopes to accurately represent the nuances and vulnerabilities of the smart grids of today.

To counter these cyber-attacks, the Cyber-Physical Resilient Energy Systems (CYPRES) models [3] a full-scale, cyber and physical power grid, which is subjected to various cyber attacks such as DDOS, ARP cache poisoning, and man-in-the-middle (MiTM) attacks. This differentiates CYPRES from other grid-security systems such as GridAttackSim, which does not incorporate physical hardware [4]. The CYPRES model’s cyber and physical response to these attacks illuminates many trends that would otherwise not be apparent when modeling either of these systems in isolation.

Much of the CYPRES system is already active in Resilient Energy Systems Lab testbed (*RESLab Testbed*) [5]. However, running a CYPRES simulation has a huge manual overhead. Each component of the system is controlled by a separate Virtual Machine (VM) in a virtualized network. Because each of these VMs must be manually started, monitored, and stopped, they form a critical bottleneck preventing researchers from testing full-scale network topologies. Another major CYPRES bottleneck is processing data once an attack is completed. To dissect the grid's response to attacks, a researcher must parse the log files of up to 7 Virtual Machines completely manually, making it very difficult to draw conclusions about the system's response to a cyber attack.

In this paper, a comprehensive testing automation infrastructure, *RESAuto*, is proposed. This infrastructure is the first automation attempt intended to handle every aspect of the cyber-physical power system simulation and emulation, from instancing the modeling programs, to collecting data, to aggregating data into a legible and actionable report, transforming the previously arduous task of running an attack into a single button press. This infrastructure will include both an automation platform that facilitates the spin-up / shutdown process of CYPRES sub-systems as well as a results reporting webapp that processes data from the previously run simulations.

The automation platform runs on a modified implementation of Jenkins [6], a CI/CD (Continuous Integration / Continuous Development) tool commonly used in the software engineering sphere. Jenkins facilitates the segmentation of multi-step tasks (referred to as Pipelines) into discrete, sequenceable steps. Jenkins is unique in its ability to run and monitor these steps on remote machines, independent of the host machine's operating system, and report the resulting error logs and artifacts back to the job sequencer. While Jenkins' original use case is for automation of software build pipelines, especially in server-less cloud infrastructure [7], many institutions, such as the Novartis Institutes for Biomedical Research, have had success with integrating Jenkins in execution of data collection tasks across distributed compute resources [8]. The Jenkins system slots seamlessly into the distributed VM infrastructure already in place for CYPRES.

Jenkins provides an extremely in-depth user interface for starting and stopping jobs. But

this level of detail is sometimes undesirable. In many cases, when running CYPRES, users simply want to see if each aspect of the simulation is already running at a glance. The other aspect of this project is a results web app shows a real-time report of the testbed's current status. Status information will be pulled from multiple sources: The Jenkins API is used to determine if the respective jobs for each sub-system are currently running, and more integration with the RTAC subsystem is planned to determine if DNP3 data generated by PowerWorld can travel through the CORE network.

CYPRES's purpose isn't just to model a power grid, it also utilizes standard intrustion detection systems (IDS) to prepare for and detect cyber intrusions. Each IDS separately implements it's own alerting system, which makes it difficult to visualize and fuse intrusion data from multiple sources. To counter this, the RESAuto monitoring web app also includes a common IDS monitoring area, which aggregates the number of IDS alerts and plots them alongside network traffic and other metrics, allowing users to easily visualize the testbed's current status. Since the testbed only allows one simulation to run at a time, knowing whether a simulation is running or not is vital to preserving the integrity of the data.

2. METHODS

2.1 System Infrastructure

2.1.1 Existing Infrastructure

The CYPRES power system modeler is built out of many different subsystems. Each of these subsystems consists of a Virtual Machine (VM) and a program that models one aspect of an interconnected power system. These VMs are managed through vSphere, which not only segments a central server into multiple virtual machines but also models the networks that connect them. Note that these virtual networks are only used to administrate the VMs. A dedicated VM with a software network modeler is used to simulate the attack network. The main subsystems modeled by CYPRES and a description of their functionality are listed below.

2.1.1.1 CORE

One of CYPRES' differentiating factors is it's cyber-physical modeling capabilities. The cyber aspects of the modern power grid are both crucial and under-studied. Power systems are often administered using networking: control centers, substations, and generators are generally on interactable subnets. This segmentation allows for unprecedented remote control, but also opens the power system to previously impossible methods of cyber-intrusion. Any full-grid defense simulator should be able to model and take into account these networking weak points.

To model these cyber aspects, CYPRES uses the open-source tool Common Open Research Emulator (CORE) [9] in a dedicated Ubuntu Linux VM to simulate network connections inside and between substations, generators, and loads in a power system. CORE is capable of not only simulating these networks, but also simulating the routers and machines in these networks. Real network packets can be sent between simulated machines, and real virtual machines can be slotted in alongside the virtual machines. CORE can also connect it's virtual network to real-world networks. In our case, the simulated network is connected to hardware RTAC devices, which function as they would in the field. This power and flexibility allows us to run real-world intrusion vectors

on a simulated power system and glean real-world data from the system's response.

2.1.1.2 PowerWorld Dynamics Studio

The major purpose of power grid networking is aggregating the data and controlling the status of headless relays in the field. Distributed Network Protocol version 3 (DNP3) data is one of the major sources of network traffic on the RESLab virtual network. This data consists of the real-time status of the thousands of generators, substations, and loads in a virtual power grid. Since this data is sent over the network, it is susceptible to attacks such as buffer overflow, man-in-the-middle, and packet sniffing. [10] To accurately model these possibilities, a cyber-physical simulator must utilize a reliable source of real-time DNP3 data.

To achieve this, CYPRES uses PowerWorld Dynamics Studio (PowerWorld DS) [11] in a dedicated Windows 10 VM to simulate the physical power system. While PowerWorld is historically used for network powerflow optimization problems, it also has a real-time dynamics simulator. Running this simulator will both emulate a real-time power grid and generate DNP3 data for multiple buses at the same time. This data is connected with the RTAC AcSELerator DNP3 Master through the CORE virtual network. [12].

2.1.1.3 RTAC

Once this data is generated, it has to be processed. Industry often uses Real-Time Automation Controllers, or RTACs, for this task. An RTAC [13], and is a specific type of programmable logic controllers that can communicate and control power system components, aggregating DNP3 data from multiple sources. Using this data, users can query this data for specific information and create automated controls or warnings that trigger directly from the data.

CYPRES implements a DNP3 master using the proprietary RTAC AcSELerator program in a dedicated Windows VM. CYPRES can also use a Python implementation of a DNP3 Master to achieve this task, but the manual overhead of using AcSELerator makes this subsystem a prime candidate for automation.

2.1.1.4 Adversary

This subsystem is often run from within the CORE VM. It implements the adversary's actions in an attempt to gain access to restricted controls over the power system. [14] MiTM, Denial of Services (DoS), and portscanning attacks have been implemented in this VM [14, 15, 16]. Attacking the RESLab testbed will trigger intrusion detection system (IDS) alerts, which are aggregated in the DataFusion subsystem.

2.1.1.5 DataFusion

This subsystem compiles data from other subsystems' monitoring software through the use of Elasticsearch, Packetbeat, Snort, Cicflowmeter, and Kibana [15]. These monitoring alerts allow us to see how a real-time balancing authority could determine if a cyber attack is taking place.

Note that this list of CYPRES subsystems is by no means all-encompassing. These subsystems have been selected for their difficulty of manual operation and importance overall.

2.1.2 Jenkins Implementation

Jenkins handles remote script execution for the automation framework, and it provides a graphical user interface (GUI) frontend for attack sequencing and remote script execution This subsection is a brief overview of the Jenkins features used by the framework.

At it's core, Jenkins consists of an interactive web-interface called the Jenkins controller. Through a webpage hosted on the Jenkins controller's machine, Jenkins provides the user with a dashboard that facilitates creating, editing, and executing jobs. The Jenkins controller also administrates the Jenkins instance: new remote executors (agents) can be added, access control can be configured, and the instance can be restarted from this frontend.

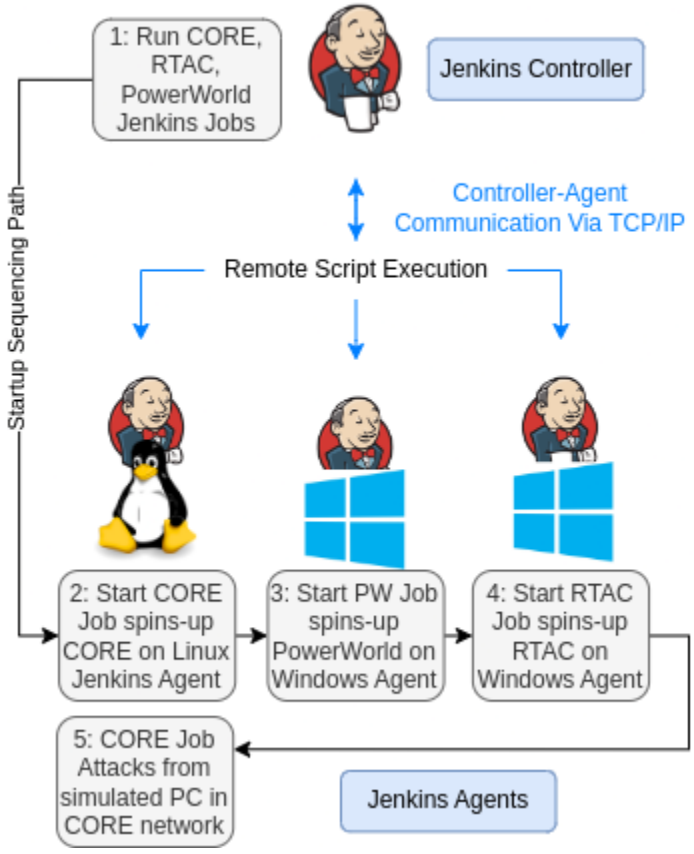

Figure 1: Jenkins based Implementation of RESAuto for RESLab Testbed

Figure 1 illustrates the pipeline of the automated process of CYPRES using Jenkins automation infrastructure. First of all, the user runs Jenkins jobs to start CORE, PowerWorld, and RTAC from the Jenkins Controller. Jenkins will then distribute these jobs to their specified agents, spinning-up each subsystem on its own dedicated machine. Once these subsystems are online, the CORE job runs attack scripts on the network through the CORE machine.

Jenkins' bread and butter is the Pipeline: a scripted sequence of commands (called steps) that split up a more complex process. Each step in a pipeline can be executed on a different Jenkins agent, and these steps can be run in series or in parallel. For example, a pipeline could consist of a step that starts the CORE subsystem, then a step that runs the attack script, and finally a step that shuts down the CORE subsystem. Each step on a pipeline is individually timed and can return

errors and log files.

The Jenkins controller also features a tool that attaches a GUI of user-configurable check-boxes and text fields to each job. The values held in these parameters can be used in the job's scripting pipeline: these variables can even be passed as arguments into the locally run file. This allows users to set program options without ever needing to touch code.

At the conclusion of a job, any generated logs (console output) are saved so that previous executions of the pipeline can be compared to current ones. Any files created by the build process can also be saved to these logs as "Artifacts." Artifacts are accessible to processes outside of Jenkins via the Jenkins application programming interface (API).

The Jenkins API also provides one more crucial function: it can be queried to display real-time job information. Information accessible through the API includes whether a job is currently running, how long it has been running, and the full console output of the spawned process. using the the API's RESTful interface, this information is accessible to external programs such as the RESAuto Dashboard.

2.2 Automation Methods

Jenkins is only used to facilitate remote execution of scripts; to sequence and automate the subsystem software locally, a local scripting language is needed. Though many subsystems in CYPRES can be controlled headlessly through use of an API, a user-interactable GUI is needed to control the subsystem once it has been initialized. As a result, RESAuto focuses on automating system startup through scripted interactions with a GUI.

Linux machines with the BASH shell are blessed with a rich scripting language out-of-the-box, and many open-source programs (such as the CORE emulator) provide functions for command-line program automation. To fill in the gaps where BASH scripts lack, such as command-line terminal user interface (TUI) interaction, transaction control languages (TCL) offshoot Expect Script is used. Windows machines are more difficult: proprietary software rarely provides an API for programmatic control of a GUI. Automating these GUI programs is made simple through use of open-source tool AutoHotkey.

2.2.1 *AutoHotkey*

AutoHotkey provides an open-source scripting language with extensive documentation centered around control of Windows-based desktop programs, and a program that scrapes window metadata from programs for more accurate scripting (WindowSpy) [17]. Scripting commands in AutoHotkey are varied, but generally use the same syntax. Nearly all commands contain a **Command**, a **Control**, and an **Argument**. These fields are explained below.

- Commands describe how to interact with an object, similar to a function in other scripting languages.
- Controls are arguments that determine which field, window, or selection to interact. This can be set to match window titles, program `.exe` file names, and myriad other distinguishing factors.
- Arguments are required supplementary information. For example, a command that auto-fills text would need to be provided the desired text as an argument.

Much of AutoHotkey's power lies in it's selector syntax. Most automation software works by sending click events and keystrokes to x-y positions on the current screen. This is inherently unstable, since windows can be resized and moved, or the resolution of the attached monitor changed. Through the use of CSS-style selectors, AutoHotkey can send events directly to the desired fields of the desired window. These selectors, called *matching groups*, are used in the background by Windows but can be scraped from windows using WindowSpy.

2.2.2 *Expect Scripting*

Expect scripts [18], at their most basic, allow a programmer to run a command line program, and conditionally react to said program's output. A program can be **Spawned**, which means it is being actively listened to. After a spawn step, the **Expect** step halts the execution of the code until the output of the spawned process matches the expect step's input string. The **Send** command inserts the supplied text directly to the terminal the expect script was first spawned. Through a

combination of these three steps, a scripted "conversation" can be formed, which can be run to automate any number of TUI programs.

2.3 Automation Implementation

The following subsections describe how the combination of Jenkins, Expect, and AutoHotkey have been applied to CYPRES to automate the manual overhead of simulating a full power grid through simplified operation.

2.3.1 Anatomy of an Attack

Before this automation framework, a user would have to remotely connect into every virtual machine in the *RESLab Testbed*, manually start and configure each sub-system, then remote into the emulated adversary's device and administer the desired attack. The automation framework splits these steps into automated, user-friendly steps. This subsection describes a new workflow, or attack anatomy.

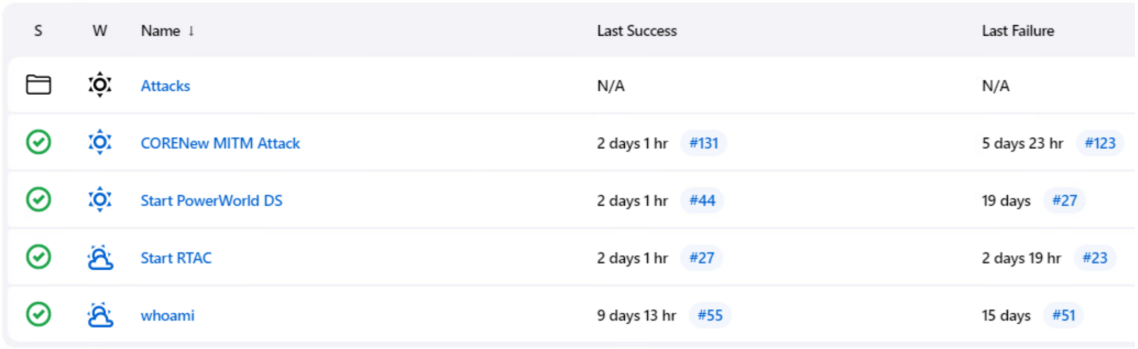

Figure 2: Jenkins Home Dashboard.

On logging into Jenkins, users are presented with the dashboard pictured in Figure 2. This dashboard has a job for each currently automated subsystem: CORE (+ the Man in the Middle attack), PowerWorld DS, and RTAC AcSELerator.

Selecting the desired subsystem and the **Build with Parameters** option brings the user to an options form. Options selected on this form change commonly-used parameters for that

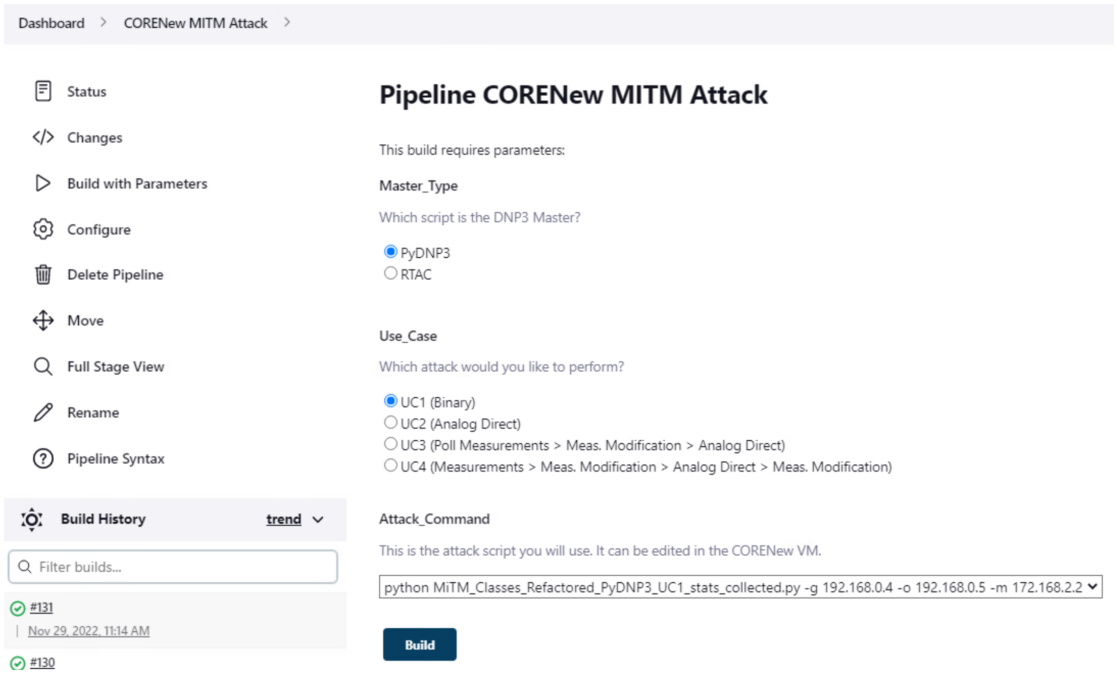

Figure 3: Job-Specific Form.

subsystem; options could include which power system case to simulate, which communication topology to emulate, whether to run specific monitoring software, or whether to defend CYPRES from attack. The form shown in Figure 3 allows the user to select options for the MiTM attack.

To spin up the desired sub-system, the user will press the **Build** button at the bottom of Figure 3. This will bring up the screen in Figure 4. Each subsystem is split into the start, confirm, and stop steps. Once the start step has concluded, Jenkins will wait for the user to click the blue box in the confirm step, which will notify Jenkins that the attack has concluded. In the time between the subsystem spin-up and this confirmation, the user will make any necessary adjustments to the subsystem and administer the cyber-attack. After the user notifies Jenkins that the attack has ended, Jenkins will spin down all the previously started programs. Logs from previous executions can be accessed from this screen as well.

2.3.2 Subsystem Details

A brief overview of each subsystem's automation implementation is provided here.

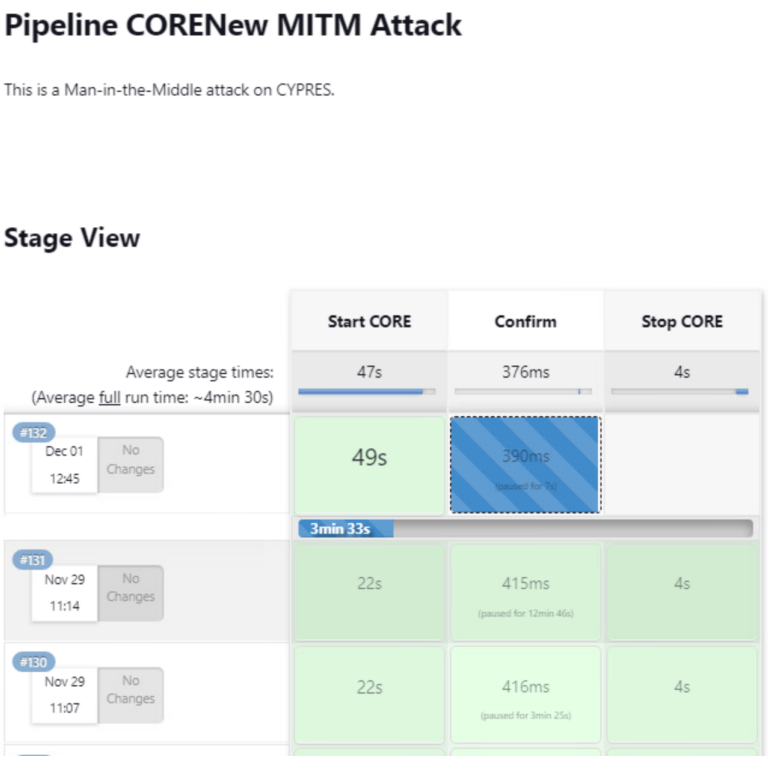

Figure 4: Job Run Screen.

2.3.2.1 CORE Subsystem

After the user runs CORE, the Jenkins job, Jenkins will call the main script in this subsystem: `startCORE.sh`. This script sequences all of the following steps, takes into account the arguments specified in Jenkins, and handles errors in the execution of starting the CORE subsystem. The first job of this script is to start the CORE GUI. The logged-in user must have `sudo` privileges, so that Jenkins can run scripts that require `sudo`. The script checks to see if `core-gui` is running: the script will fail gracefully if a simulation is already taking place. Then, it starts the CORE daemon that the GUI will connect with. Starting the `core-gui` program itself is a little more complicated. The GUI can take command-line arguments, so the script pipes in the path to the network topology file selected in Jenkins. To account for `core-gui`'s tendency to crash without an error code and require a restart, the process is spawned through an expect script, which fails unless the `core-gui` program comes online within a certain timeout.

Monitoring programs, including Packetbeat, Elasticsearch, Logstash, and Kibana, are all started locally after `core-gui` is started. Certain monitoring software, such as Snort and Wireshark, must be run in the virtual router on the CORE network. The router has a static IP address and is accessed through Secure Shell (SSH). Because it is spun up at runtime by CORE, SSH keys can not be uploaded to the machines, forcing users to interact with SSH shell logins to start this software. To get around this limitation, an expect script is again used to automate the SSH login interaction and execution. This script forwards the Wireshark GUI back to the CORE VM to be interacted with, and starts Snort, a background network traffic analyzer.

2.3.2.2 Man-in-the-Middle Attack

Since the MiTM attack only relies on the CORE subsystem, it has been implemented into the CORE Jenkins job. When the CORE job is selected, a GUI for the MiTM attack appears. After CORE is fully spun up, Jenkins will create a terminal, already running in the adversary's machine, with the attack script command already typed in. To attack the power system, the user may choose to modify the attack, or simply hit enter and begin collecting data.

2.3.2.3 PowerWorld DS Subsystem

Once the real-time power system simulation in PowerWorld DS is online, the buses (substations) in the case act as DNP3 Outstations that generate DNP3 packets containing power system data of the system [19, 20]. The PowerWorld DS runs on a Windows machine. This program is GUI-only for our applications: as such, automating it requires AutoHotkey (AHK).

After Jenkins starts the correct PowerWorld version, the `WinMenuSelectItem` command is used to open the correct power system case. These files are provided by argument to the AHK script, and can be sent and modified from Jenkins. This process is completed twice, once for the power system case and once for the one-line diagram. The AHK script then uses `WinMenuSelectItem` to start the server, and simulation. The script will exit at the conclusion of this step. To stop PowerWorld, the AHK script will first pause the simulation, then abort. Doing so avoids a bug in PowerWorld that causes the program to hang indefinitely.

2.3.2.4 RTAC AcSELerator

CYPRES's RTAC subsystem is functioned as a data concentrator that collects data and sends control commands to the DNP3 outstations hosted in PowerWorld DS [10]. RTAC's control over these vital power system functions makes it a desirable target for an adversary: an attack chain can end with a command to trip a relay and influence the physical reliability of the power system.

The RTAC subsystem is entirely implemented within the proprietary program RTAC AcSELerator to load the configuration file. Similar to PowerWorld DS, AcSELerator has an API, but it cannot be used due to the necessity of manual operation once the subsystem has been spun up. Once the Jenkins job for RTAC is executed, Jenkins runs the automation script. This script opens up RTAC AcSELerator and waits for the program to load. The script will then auto-fill the correct key in the first of two password protect screens. Once the password has been entered, the desired RTAC topology file must be selected from a table. The individual cells in the table do not possess AutoHotkey selectors, and will only respond to keyboard and mouse inputs. To select the desired file, the AHK script resorts to sending click events to x and y coordinates. While this would normally be an unreliable method of control, the script resizes the window to a known size and uses coordinates relative to the window's top corner. This ensures that the control will always be in the same place and will always receive the click event. After passing through the second password protect screen, the RTAC is online and can interact with PowerWorld DS and CORE (if those subsystems are also running).

2.3.3 Status Web App

Due to the distributed nature of the RESLab architecture, it is difficult to determine which subsystems are running at any given time. Critical information sources, such as error logs, are split up across three virtual machines, making debugging difficult. Additionally, large amounts of data generated by multiple intrusion detection systems (IDS) are tedious to manually sift through. The status webapp solves these problems by aggregating real-time-status and intrusion detection data in one place, to provide a testbed "heartbeat".

2.3.3.1 Jenkins Status Updates

Jenkins naturally provides the tools to solve many of the above visibility problems. When sub-systems are run from Jenkins, job status information such as job start time, predicted runtime, and spawned process output are already accessible through the Jenkins frontend. However, this information is often hidden in sub menus, or in extreme cases, not visible to the user at all. Because of this, the RESAuto dashboard uses Jenkins' REST API to periodically query the status of the three main jobs. The results of this query is a JSON document that contains all of the above information and more. By parsing this JSON, the current status of the testbed can be determined. The refresh interval is defined as one second: any less resolution, and the information runs the risk of being inaccurate, and not real-time.

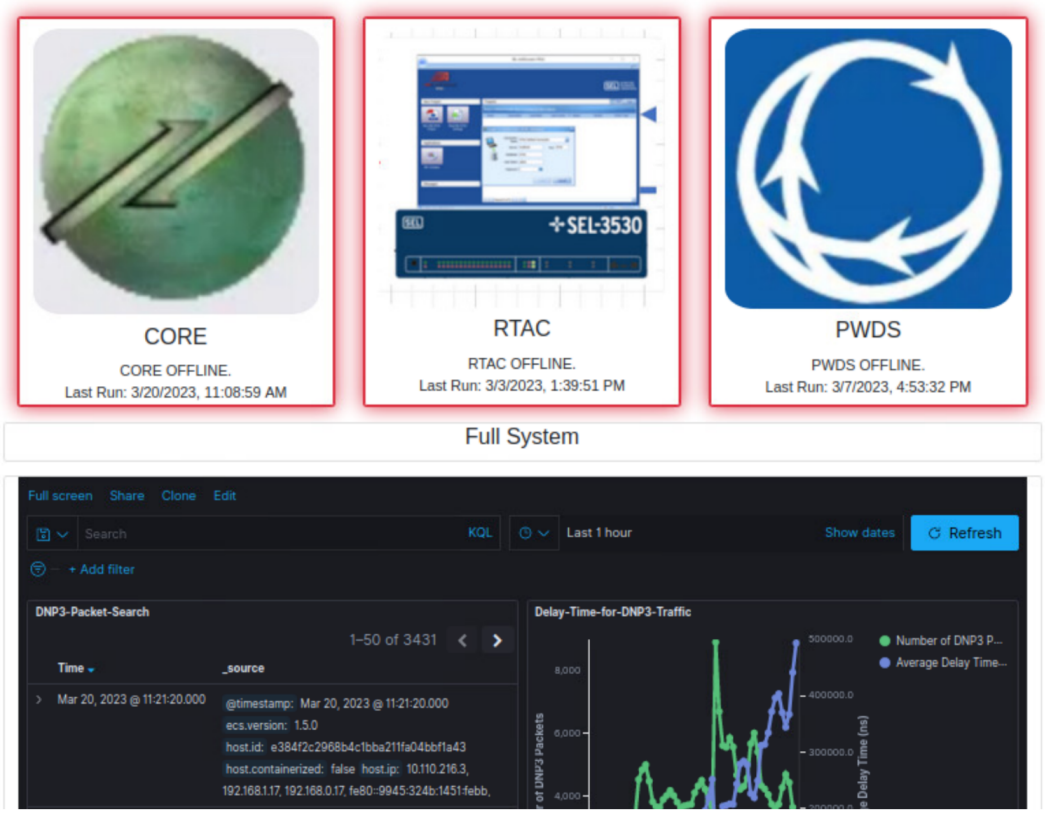

Figure 5: Real-Time Dashboard

Figure 5 shows the look and feel of the dashboard. When offline, the subsystem tiles have a red border and display the last time they were online. When online, the tiles have a green border and display how long they've been running for. Underneath these tiles are the SNORT and Packetbeat logs.

2.3.3.2 SNORT Intrusion Detection and Packetbeat Network Monitoring

One of RESLab's major intrusion detection systems is SNORT. SNORT is a program that runs on any router that has the possibility of being compromised in a cyber attack. It allows users to define "rules", or filters that are applied to network traffic. If the real-time network traffic matches activity defined in one of these rules, that activity is determined to be notable. Activity detection causes an alert to be posted to a local log file, which can later be analyzed for future intrusion prevention. SNORT notoriously generates a large amount of alerts, especially when a real cyber-intrusion is being administered. To counter this, RESLab has an ELK stack that eases the difficulty of visualizing this data. ELK stands for Elasticsearch, Logstash, and Kibana: each of which are open-source logging tools. A quick summary of their usage is found below:

- Elasticsearch [21]: The database backend of the ELK stack. At it's core, Elasticsearch is a NoSQL database with built-in indexing. These properties make it very useful for storing schema-less data such as log messages. Indexing is additionally helpful for locating log messages with specific filters, such as finding logs within a certain timeframe or looking for messages that contain the attacker's IP address.

- Logstash [21]: The client-side logging application. Logstash monitors a local log file (like the one generated by SNORT) for changes. When changes occur, Logstash assigns a timestamp to those changes and pushes a copy to the ElasticSearch database. Logstash also handles the local machine's storage: if there isn't enough storage on the machine, Logstash puts the Elasticsearch index into read-only mode.

- Kibana [21]: The visualization side of the stack. Kibana takes groups of data and visualizes changes in them. Kibana is most commonly used in this project to visualize the number of

SNORT alerts before and after a cyber-intrusion.

- Packetbeat [21]: A plugin that generates data for the ELK stack. It describes itself as "a distributed real-time wireshark". It's primarily used in the testbed for overall metrics on internet traffic: for example, increased network traffic can signal that a DOS attack is taking place.

Previous implementations of Datafusion use the full Kibana dashboard to visualize many different aspects of the packetbeat data, such as packet errors over time or TLS Handshake latency. However, since this is meant to be an at-a-glance dashboard, the dashboard only displays alert message volume over the last hour for both SNORT and Packetbeat. Kibana allows the individual dashboard elements to be copied and embedded in other webpages, so the dashboard directly links to Kibana to render the dashboard elements.

Figure 6: SNORT and Packetbeat Dashboard

Figure 6 shows various metrics derived from the SNORT and Packetbeat logs. These include DNP3 traffic delay time, SNORT alert type, SNORT alert frequency, and raw DNP3 data. These metrics will be used to quickly determine if the CYPRES network is running, and if it is under attack.

3. RESULTS

3.1 CORE Results

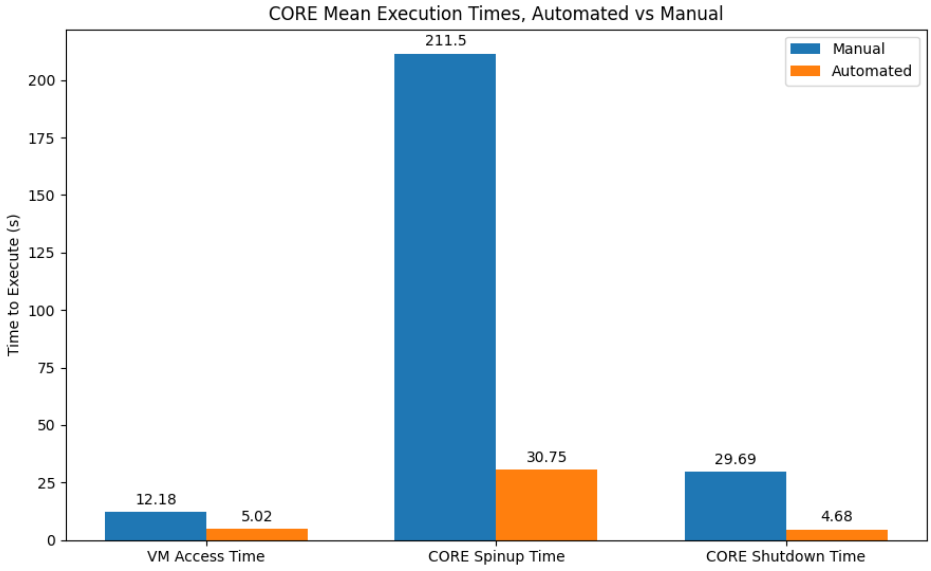

Figure 7: CORE Time to Start

In timed trials, CORE took at least 253 total seconds to spin up. The average number may even be higher, since the timed user was familiar with the process of quickly running these steps. This gives an improvement of 635% as shown in Figure 7. Additionally, Jenkins allows these steps to be run in parallel with the startup of other subsystems, further compounding time savings. The automation also handles many edge cases that trip up users. For example, if a user tries to start the CORE subsystem while it is already running, the networks would interfere with each other and the operation of CYPRES. The automation scripts handle this by notifying the user and not allowing the script to continue.

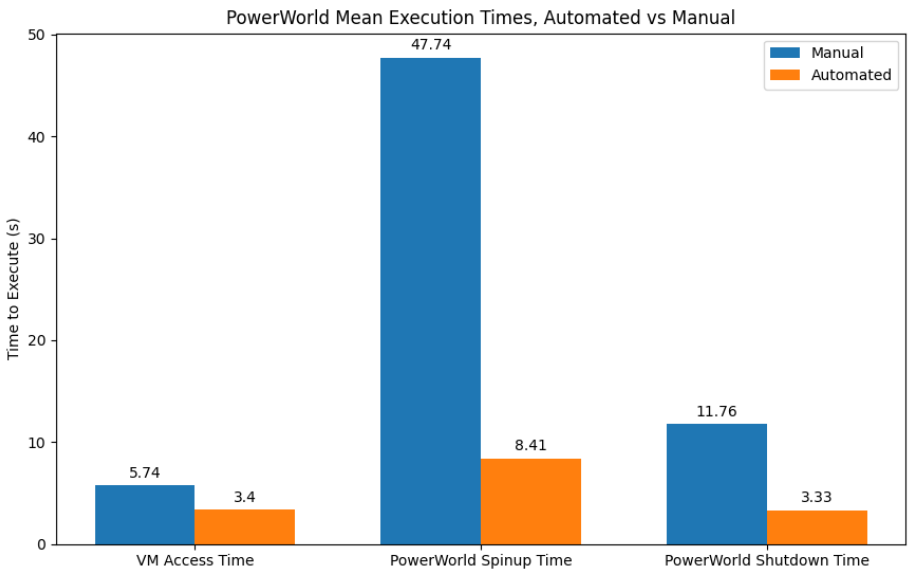

Figure 8: PowerWorld Time to Start

3.2 PowerWorld Results

In timed trials, PowerWorld DS took 65 seconds to spin up. Automation trials took only 15 seconds, an improvement of 433% (Figure 8). As with CORE, these time savings are compounded considering that many sub-systems can be spun up in parallel. The PowerWorld sub-system also handles the edge case of multiple programs running at once. The Jenkins job will fail if a user tries to start a new PowerWorld instance while one is already running.

3.3 RTAC Results

In timed trials, RTAC AcSELerator took 120 seconds to spin up. Automation trials took 99 seconds, an improvement of around 120% (Figure 9). While this improvement initially seems small, the main pain point of RTAC is the heavy manual interaction of starting the subsystem. With two password prompts, the user may not be familiar with the passwords and need to spend even more time searching documentation for them.

The RTAC subsystem can also be used to validate the full system. In the RTAC controller menu, one can check the connection diagnostics of RTAC (Figure 10). Two fields are of interest

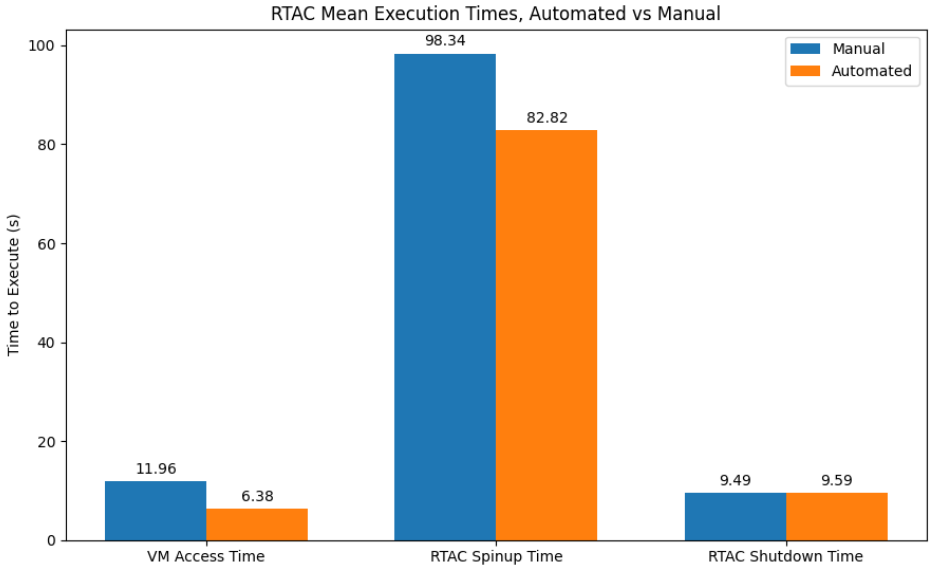

Figure 9: RTAC Time to Start

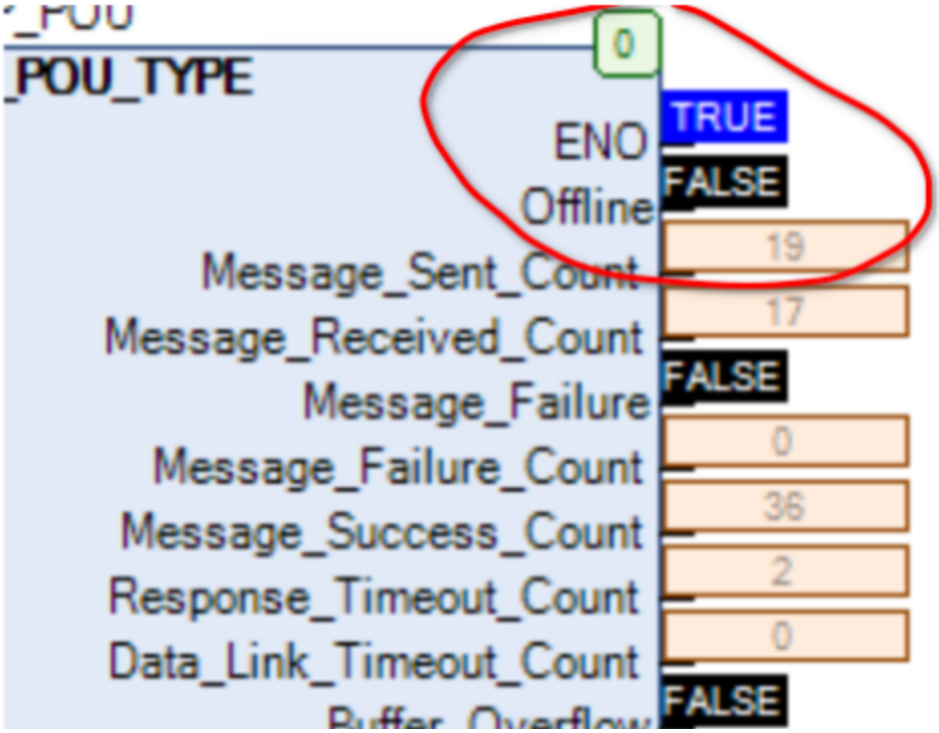

Figure 10: RTAC System Verification

here. If the `Offline` field reads FALSE, then the RTAC is connected to the CORE virtual network. If the `Message_Success_Count` counter continues to increase, the RTAC is connected to the PowerWorld DNP3 Outstations. With these two fields, the functionality of the network can be confirmed.

4. CONCLUSION

4.1 Conclusion and Future Work

Modeling complex power systems often necessitates a massive manual overhead. In this paper, I propose RESAuto, a common automation framework to increase speed and facilitate usage of the CYPRES system. The effectiveness of RESAuto is shown by automating three subsystems that previously required extensive amounts of time and effort to run. RESAuto decreased the time spent initializing these subsystems by up to 635% and reduced the manual interaction required to simply pressing a single button. It also drastically eased the troubleshooting and visualization of the CYPRES framework by introducing the CYPRES Dashboard, allowing users to see each subsystem's status and packet data in real-time. Both of these systems not only decrease the workload of experienced users, but also abstract the non-essential technical details from new users who don't have extensive experience in either the cyber or physical field of power grids.

4.1.1 Scalability

RESLab is unique in how realistically it models a power grid: this realism means that many different types of cyber-attacks can be attempted. The same attack can often have different parameters, or ways in which the attack can be run. Jenkins was selected as the distributed computing solution primarily because of it's scalability to this style of testbed framework. As mentioned earlier, Jenkins Parameters are options that users select before running jobs. These parameters are already richly integrated into the testbed: they control attack options like how strong the attack is, whether to use monitoring software, or which sub-systems the program can expect to be running at the time of the attack. These parameters, crucially, can be edited by users, allowing old jobs to be refactored to implement new cyber-attacks. For example, the CORE job currently runs a man-in-the-middle attack, but by editing the parameter list, it can be configured to run an arp cache poisoning attack instead. To assist with user-editing, a large knowledge base of Jenkins information was compiled during this project to show future users how to integrate their testcases into the

RESAuto framework.

While RESAuto is intellectually scalable, it is also physically scalable as well. Jenkins' support of multiple agents means that future subsystems can be automated as well, even if they are on separate machines. RESAuto isn't even limited to one script per agent: multiple scripts can be run in parallel even on the same machine using Jenkins. Larger pipelines can be built that spin up each sub-system in order, making RESAuto a truly one-click solution for cyber-physical testbed simulation.

4.1.2 Future Work

There are plenty of avenues that this project could take in the future. With the results-tracking webapp and Jenkins, tests can be orchestrated with just a few button presses. But Jenkins' history as a software CI/CD tool opens even more doors. As it stands, the testbed is currently being used as a development environment as well as a testing site. This introduces issues where many of the same virtual machines have been cloned so each student can work on their own projects without interfering with each other. If connected to a Git repository, Jenkins can be configured to automatically run a test suite on newly pushed code. By utilizing these Git hooks, Jenkins could make it easier for students to develop their code locally and test it automatically on the testbed.

At the extreme end, Jenkins could even be further integrated into the testbed's infrastructure as well. Using tools like Minimega, Jenkins could deploy the entire testbed at the click of a button. It could then connect to automatically spawned agents within the testbed, and orchestrate tests from there. Doing so would greatly improve the reproducibility of the test suite, since each test is completely segmented from other tests.

REFERENCES

[1] I. Colak, "Introduction to smart grid," in *2016 International Smart Grid Workshop and Certificate Program (ISGWCP)*, pp. 1–5, IEEE, 2016.

[2] C. John, B. Ramachandran, and E. Kalaimannan, "Impact of targeted cyber attacks on electrical power systems," in *National Cyber Summit*, pp. 278–292, Springer, 2019.

[3] "Cyber Physical Resilient Energy Systems (CYPRES)." `https://cypres.engr.tamu.edu/`.

[4] T. D. Le, A. Anwar, S. W. Loke, R. Beuran, and Y. Tan, "Gridattacksim: A cyber attack simulation framework for smart grids," *Electronics*, vol. 9, no. 8, p. 1218, 2020.

[5] A. Sahu, P. Wlazlo, Z. Mao, H. Huang, A. Goulart, K. Davis, and S. Zonouz, "Design and evaluation of a cyber-physical testbed for improving attack resilience of power systems," *IET Cyber-Physical Systems: Theory & Applications*, vol. 6, no. 4, pp. 208–227, 2021.

[6] "Jenkins," 2022. `https://www.jenkins.io/`.

[7] S. Sinde, B. Thakkalapally, M. Ramidi, and S. Veeramalla, "Continuous integration and deployment automation in aws cloud infrastructure," *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, pp. 1305–1309, 06 2022.

[8] I. K. Moutsatsos, I. Hossain, C. Agarinis, F. Harbinski, Y. Abraham, L. Dobler, X. Zhang, C. J. Wilson, J. L. Jenkins, N. Holway, *et al.*, "Jenkins-ci, an open-source continuous integration system, as a scientific data and image-processing platform," *SLAS DISCOVERY: Advancing Life Sciences R&D*, vol. 22, no. 3, pp. 238–249, 2017.

[9] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *MILCOM 2008-2008 IEEE Military Communications Conference*, pp. 1–7, IEEE, 2008.

[10] H. Huang, C. M. Davis, and K. R. Davis, "Real-time power system simulation with hardware devices through dnp3 in cyber-physical testbed," in *2021 IEEE Texas Power and Energy Conference (TPEC)*, pp. 1–6, IEEE, 2021.

[11] "PowerWorld Simulator," 2020. https://www.powerworld.com/.

[12] I. S. Association *et al.*, "Ieee standard for electric power systems communications-distributed network protocol (dnp3)," *IEEE: Piscataway, NJ, USA*, 2014.

[13] D. Sheet-SEL, "3555 real time automation controller (rtac)," *Accessed: Aug*, vol. 3, pp. 20181231–170401, 2019.

[14] P. Wlazlo, A. Sahu, Z. Mao, H. Huang, A. Goulart, K. Davis, and S. Zonouz, "Man-in-the-middle attacks and defence in a power system cyber-physical testbed," *IET Cyber-Physical Systems: Theory & Applications*, vol. 6, no. 3, pp. 164–177, 2021.

[15] A. Sahu, Z. Mao, P. Wlazlo, H. Huang, K. Davis, A. Goulart, and S. Zonouz, "Multi-source multi-domain data fusion for cyberattack detection in power systems," *IEEE Access*, vol. 9, pp. 119118–119138, 2021.

[16] T. Tarman, T. Rollins, L. Swiler, J. Cruz, E. Vugrin, H. Huang, A. Sahu, P. Wlazlo, A. Goulart, and K. Davis, "Comparing reproduced cyber experimentation studies across different emulation testbeds," in *Cyber Security Experimentation and Test Workshop*, pp. 63–71, 2021.

[17] "AutoHotkey," 2022. https://www.autohotkey.com/.

[18] "Expect Scripts," 2022. https://linux.die.net/man/1/expect.

[19] T. J. Overbye, Z. Mao, K. S. Shetye, and J. D. Weber, "An interactive, extensible environment for power system simulation on the pmu time frame with a cyber security application," in *2017 IEEE Texas Power and Energy Conference (TPEC)*, pp. 1–6, IEEE, 2017.

[20] T. J. Overbye, Z. Mao, A. Birchfield, J. D. Weber, and M. Davis, "An interactive, stand-alone and multi-user power system simulator for the pmu time frame," in *2019 IEEE Texas Power and Energy Conference (TPEC)*, pp. 1–6, IEEE, 2019.

[21] "Elasticsearch," 2022. https://www.elastic.co/elastic-stack/.