2023 IEEE Texas Power and Energy Conference (TPEC) | 978-1-6654-9071-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/TPEC56611.2023.10078723

A Common Automation Framework for Cyber-Physical Power System Studies

Ethan Cope, Hao Huang, *Member, IEEE,* Abhijeet Sahu, *Member, IEEE,* Katherine Davis, *Senior Member, IEEE*
Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA
Email: ethancope@tamu.edu, hao_huang@tamu.edu, abhijeet_ntpc@tamu.edu, katedavis@tamu.edu

***Abstract*—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) introduces a cyber-physical energy management system for end-to-end defense whose design involves 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 subsystems of the framework involve interdependencies across different applications. This paper details a common automation framework for CYPRES that both removes this manual overhead from running the system and drastically improves CYPRES's initialization speed. With Jenkins, a continuous integration/continuous development tool, the authors have established this automation infrastructure, *RESAuto*, for CYPRES system and demonstrated the benefits and effectiveness of automating manually intensive operations with simplified operations and less time consumption. This automation system makes it easier to test more complex threat scenarios for larger, more realistic, and manually intensive scenarios.**

***Index Terms*—modeling, simulation, automation, virtualization**

I. 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, decouple 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.

978-1-6654-9071-9/23/$31.00 ©2023 IEEE

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.

II. System Infrastructure

A. Existing CYPRES Infrastructure

The CYPRES power system modeler is formed of many different subsystems. Each of these subsystems consists of a 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 software modeler is used to simulate the attack network. The main subsystems modeled by CYPRES and a brief description of their functionality are listed below.

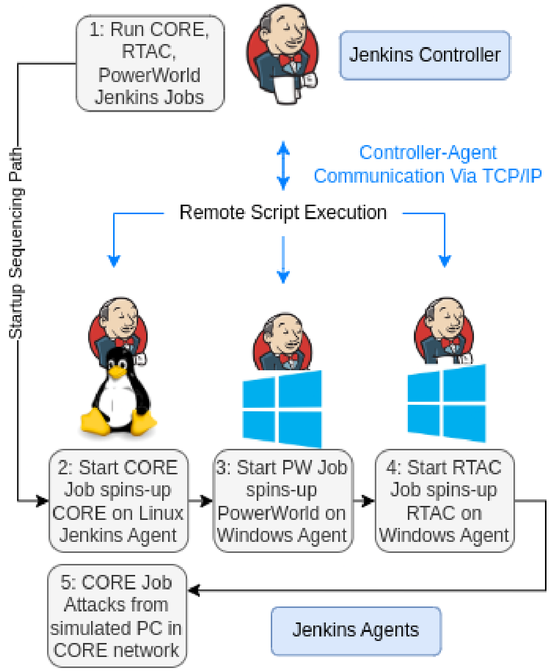
- CORE: This subsystem uses the open-source tool Common Open Research Emulator (CORE) [9] in a dedicated Ubuntu Linux VM to simulate network connections inside and between substations, generators, and loads in a power system. It also models routers, switches and firewalls inside of the network itself.
- RTAC: RTAC stands for Real-Time Automation Controller [10], and refers to a specific type of programmable logic controllers that can communicate and control power system components. This subsystem 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.
- PowerWorld Dynamics Studio: This subsystem uses PowerWorld Dynamics Studio (PowerWorld DS) [11] in a dedicated Windows 10 VM to simulate the physical power system. It connects with the RTAC AcSELerator DNP3 Master over the CORE virtual network with Distributed Network Protocol version 3 (DNP3) [12].
- Adversary [13]: 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. MiTM, Denial of Services (DoS), and portscanning attacks have been implemented in this VM [13]–[15].
- DataFusion: This subsystem compiles data from other subsystems' monitoring software through the use of Elasticsearch, Packetbeat, Snort, Cicflowmeter, and Kibana [14].

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.

B. 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 section 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 can be added, access control can be configured, and the instance can be restarted from this frontend.



1: Run CORE, RTAC, PowerWorld Jenkins Jobs
Jenkins Controller
Controller-Agent Communication Via TCP/IP
Startup Sequencing Path
Remote Script Execution
2: Start CORE Job spins-up CORE on Linux Jenkins Agent
3: Start PW Job spins-up PowerWorld on Windows Agent
4: Start RTAC Job spins-up RTAC on Windows Agent
5: CORE Job Attacks from simulated PC in CORE network
Jenkins Agents

Fig. 1. Jenkins based Implementation of RESAuto for *RESLab Testbed*

Jenkins' suitability to this project becomes apparent with the introduction of the Jenkins Agent. A Jenkins agent, or node, is a computer that can be remotely controlled by Jenkins. Any computer connected to the same network as the Jenkins controller can be an agent, regardless of operating system, as long as it is actively running the agent script provided by the Jenkins controller. Agents usually have some form of automation script that a user would like to run remotely. The Jenkins controller is used to remotely invoke a local execution of this script, meaning that a program invoked by Jenkins operates as if a user had executed it manually.

Fig. 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 checkboxes 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 arguments without ever touching code.

At the conclusion of a job, these logs 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).

III. 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 each subsystem in CYPRES can be controlled headlessly through an API, an 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.

A. 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) [16]. 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.

Part 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.

B. Expect Scripting

Expect scripts [17], 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.

IV. Automation Implementation

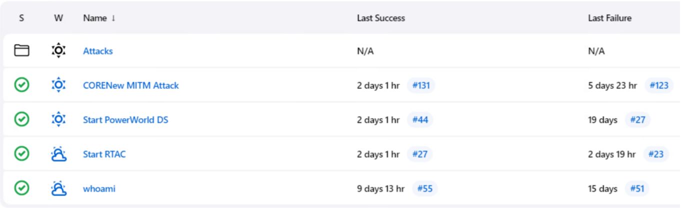
The following sections describe how the combination of Jenkins, Expect, and AutoHotkey have been applied to CYPRES to automate the manual overhead of simulating a full power grid through simplified operation.

A. 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 section describes a new workflow, or attack anatomy.

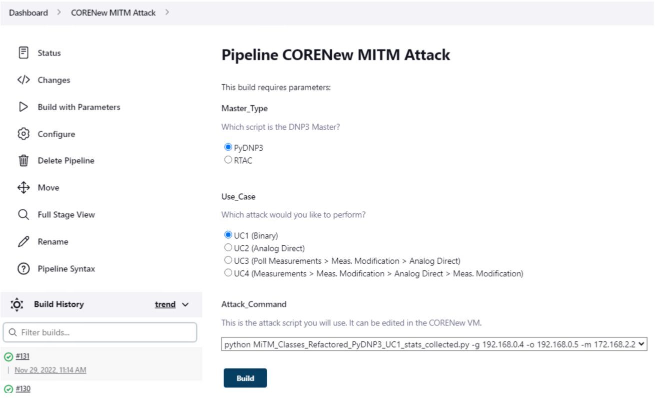
On logging into Jenkins, users are presented with the dashboard pictured in Fig 2. This dashboard has a job for each currently automated subsystem: CORE (+ the Man in the Middle attack), PowerWorld DS, and RTAC AcSELerator.

Selecting the desired subsystem and the **Build with Parameters** option brings the user to an options form. Options



S	W	Name ↓	Last Success	Last Failure
		Attacks	N/A	N/A
✓		CORENew MITM Attack	2 days 1 hr #131	5 days 23 hr #123
✓		Start PowerWorld DS	2 days 1 hr #44	19 days #27
✓		Start RTAC	2 days 1 hr #27	2 days 19 hr #23
✓		whoami	9 days 13 hr #55	15 days #51

Fig. 2. Jenkins Home Dashboard.



Dashboard > CORENew MITM Attack >

#131
Nov 29, 2022, 11:14 AM
#130

Pipeline CORENew MITM Attack

This build requires parameters:

Master_Type

Which script is the DNP3 Master?

● PyDNP3
○ RTAC

Use_Case

Which attack would you like to perform?

● UC1 (Binary)
○ UC2 (Analog Direct)
○ UC3 (Poll Measurements > Meas. Modification > Analog Direct)
○ UC4 (Measurements > Meas. Modification > Analog Direct > Meas. Modification)

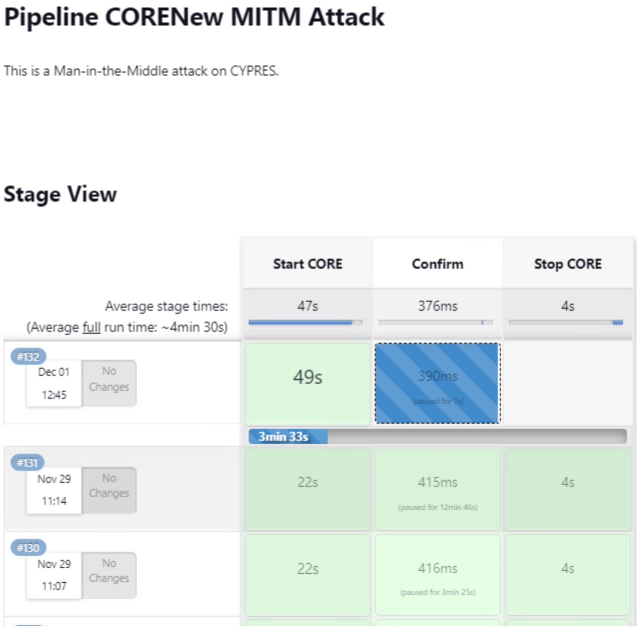
Attack_Command

This is the attack script you will use. It can be edited in the CORENew VM.

python MiTM_Classes_Refactored_PyDNP3_UC1_stats_collected.py -g 192.168.0.4 -o 192.168.0.5 -m 172.168.2.2

Fig. 3. Job-Specific Form.

selected on this form change commonly-used parameters for that 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 Fig. 3 allows the user to select options for the MiTM attack.



Pipeline CORENew MITM Attack

This is a Man-in-the-Middle attack on CYPRES.

Stage View

	Start CORE	Confirm	Stop CORE
Average stage times: (Average full run time: ~4min 30s)	47s	376ms	4s
#132 Dec 01 12:45 No Changes	49s	390ms (paused for 7s)	
	3min 33s		
#131 Nov 29 11:14 No Changes	22s	415ms (paused for 12min 46s)	4s
#130 Nov 29 11:07 No Changes	22s	416ms (paused for 3min 25s)	4s

Fig. 4. Job Run Screen.

To spin up the desired sub-system, the user will press the **Build** button at the bottom of Fig. 3. This will bring up the screen in Fig. 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.

B. Subsystem Details

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

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) 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.

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 [18], [19]. 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 AHK script uses the `WinMenuSelectItem` command 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.

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 [20]. 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).

V. Results

A. CORE Results

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 5. Additionally, Jenkins allows these steps to be run in paralell 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.

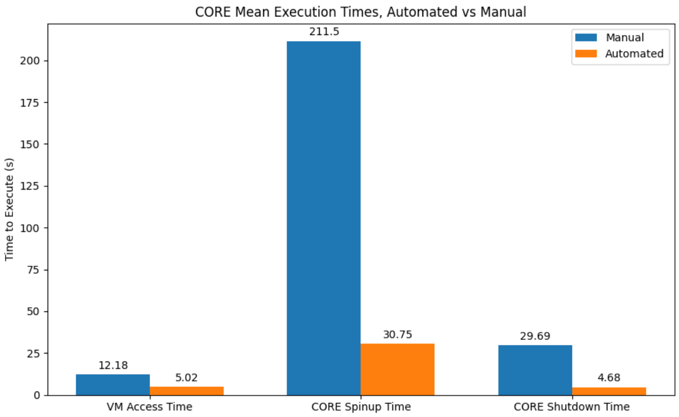


Fig. 5. CORE Time to Start

B. PowerWorld Results

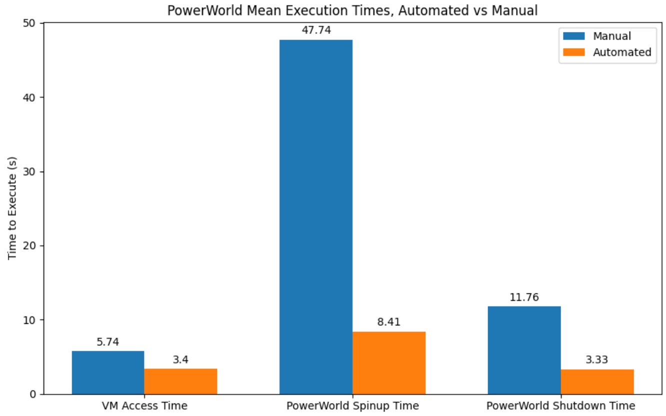


Fig. 6. PowerWorld Time to Start

In timed trials, PowerWorld DS took 65 seconds to spin up. Automation trials took only 15 seconds, an improvement of 433% (Fig. 6). 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.

C. RTAC Results

In timed trials, RTAC AcSELerator took 120 seconds to spin up. Automation trials took 99 seconds, an improvement of around 120% (Fig. 7). 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

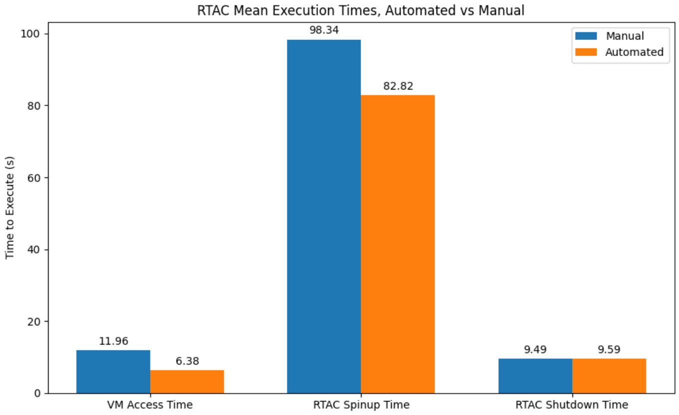


Fig. 7. RTAC Time to Start

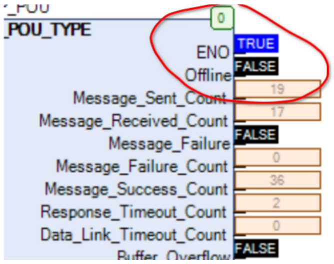

_POU
_POU_TYPE
0
ENO TRUE
Offline FALSE
Message_Sent_Count 19
Message_Received_Count 17
Message_Failure FALSE
Message_Failure_Count 0
Message_Success_Count 36
Response_Timeout_Count 2
Data_Link_Timeout_Count 0
Buffer_Overflow FALSE

Fig. 8. RTAC System Verification

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 (Fig. 8). Two fields are of interest 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.

VI. Conclusion and Future Work

Modeling complex power systems often necessitates a massive manual overhead. In this paper, we 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. Future work on this project will include the integration of a results-processing website, further extending the library of attacks that can be automated, and expanding further on integration with RTAC that will allow Jenkins-specified RTAC files to be selected.

Acknowledgment

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.

References

[1] I. Colak, "Introduction to smart grid," in *2016 International Smart Grid Workshop and Certificate Program (ISGWCP)*. IEEE, 2016, pp. 1–5.
[2] C. John, B. Ramachandran, and E. Kalaimannan, "Impact of targeted cyber attacks on electrical power systems," in *National Cyber Summit*. Springer, 2019, pp. 278–292.
[3] "Cyber Physical Resilient Energy Systems (CYPRES)." [Online]. Available: 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. [Online]. Available: 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*. IEEE, 2008, pp. 1–7.
[10] D. Sheet-SEL, "3555 real time automation controller (rtac)," *Accessed: Aug*, vol. 3, pp. 20 181 231–170 401, 2019.
[11] "PowerWorld Simulator," 2020. [Online]. Available: https://www.powerworld.com/
[12] IEEE Standards Association, "IEEE standard for electric power systems communications-distributed network protocol (dnp3)," *IEEE*, Piscataway, NJ, USA, 2014.
[13] 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.
[14] 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. 119 118–119 138, 2021.
[15] 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*, 2021, pp. 63–71.
[16] "AutoHotkey," 2022. [Online]. Available: https://www.autohotkey.com/
[17] "Expect Scripts," 2022. [Online]. Available: https://linux.die.net/man/1/expect
[18] 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)*. IEEE, 2017, pp. 1–6.
[19] 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)*. IEEE, 2019, pp. 1–6.
[20] 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)*. IEEE, 2021, pp. 1–6.