RTDS-RSCAD Logic Bomb – Proof of Concept

Daniele Ricciardelli
Electric and Computer Engineering
California State Polytechnic University (Cal Poly Pomona)
Pomona, USA
ricciardelli@cpp.edu

***Abstract*— This paper presents a cybersecurity project designed around the concept of a logic bomb, implemented in Python for educational purposes. The project entails developing a stealthy script capable of activating at a predetermined time to perform controlled file manipulations, specifically targeting critical files required to build a smart grid in the RSCAD software. The primary focus is on creating a harmless demonstration that illustrates key cybersecurity concepts including stealth execution, persistence mechanisms, conditional activation, and self-deletion capabilities. The script achieves persistence through registration as a disguised Windows service, embedding itself within system startup processes, thereby evading standard detection methods. Upon activation, the script executes predefined actions, such as modifying or deleting selected files, to simulate a realistic cybersecurity threat scenario in a controlled, ethical environment. Moreover, if desired, the script can erase its own traces, deleting itself entirely after completing its intended actions. This educational tool provides students with practical insights into detecting, understanding, and mitigating potential cyber threats.**

***Keywords* — Cybersecurity, Logic Bomb, Python Scripting, Stealth Execution, Conditional Activation, Port-Scanning, Smart Grid, RSCAD Software, Ethical Hacking, File Manipulation, Persistence Mechanism, Self-Deletion, Windows Registry, PowerShell, Command Prompt, PyInstaller**

I. Introduction

In recent years, the increased integration of real-time digital simulation (RTDS) systems, such as those employing RSCAD software, into the operational infrastructure of smart grids has introduced significant cybersecurity concerns. At Southern California Edison (SCE), electrical engineering teams regularly utilize RTDS machines to simulate grid behavior, evaluate infrastructure resilience, and prepare for real-world deployments. However, through informal interviews conducted with current student interns at SCE, it became apparent that security protocols surrounding critical simulation files and RTDS machinery access are insufficiently rigorous. This environment leaves sensitive equipment vulnerable to potential cyber threats, particularly those originating from compromised devices

II. Thinking Process

A. Objective

The primary objective of this project is to bridge foundational knowledge in computer and electrical engineering with fundamental cybersecurity practices, specifically highlighting vulnerabilities within RTDS environments that utilize RSCAD software. By creating a controlled cybersecurity demonstration, this project aims to raise awareness among engineers and stakeholders about how easily critical infrastructure can become compromised through seemingly benign cybersecurity oversights.

B. Project Scope and Methodology

Initially, several potential attack vectors targeting SQL databases associated with RSCAD software were examined, revealing multiple vulnerabilities that could be exploited given direct physical access to the RTDS hardware. Although effective, this approach was deemed impractical due to the limited likelihood of an attacker gaining direct physical access without detection.

Subsequently, attempts to directly exploit vulnerabilities within the RTDS hardware itself were considered. However, such attacks posed significant risks, potentially leading to irreversible damage or operational disruptions, thereby negatively impacting ongoing research and academic usage.

C. Chosen Approach: Logic Bomb Implementation

To address these constraints, the chosen strategy involves developing a logic bomb; a stealthy, self-replicating Python script, targeting the host computers used to interact with the RTDS equipment rather than directly attacking the hardware itself. This approach allows the demonstration of cybersecurity vulnerabilities without physically compromising the RTDS machine. By embedding itself in the Windows registry and masquerading as a legitimate service, the script maintains persistence and stealthily executes predefined, harmless actions at a specified trigger time. Furthermore, the script is designed with a self-deletion mechanism, allowing it to remove all traces of its existence after execution, underscoring the difficulties associated with tracing and mitigating similar threats.

D. Educational and Ethical Considerations

Ultimately, this project serves as an educational tool designed to illustrate critical cybersecurity concepts including stealth execution, persistence mechanisms, conditional activation, and self-deletion. It aims to foster greater understanding among electrical engineering students and professionals at SCE regarding the cybersecurity vulnerabilities present in their operational environments. By demonstrating the ease with which an individual possessing basic cybersecurity knowledge can compromise system integrity, this project emphasizes the urgent need for enhanced security protocols and increased cybersecurity awareness.

III. Tools and Environment

A. Hardware

This project was specifically designed for implementation within a Windows environment, given its objective to alter the operational parameters within the RSCAD software. Therefore, a Windows-compatible machine capable of running executable applications was essential. To verify real-time communication and simulate realistic conditions, an Ethernet-to-USB cable was acquired, enabling efficient monitoring of port activities. In absence of such specialized hardware, the standard Ethernet connection typically available with RTDS hardware could alternatively serve the same purpose.

A critical aspect of ensuring the realism of this cybersecurity demonstration was establishing a functional connection between the compromised host computer and the Real-Time Digital Simulator (RTDS) machine. The RTDS machine is an advanced real-time simulation platform widely used to test and validate smart grid infrastructure, control strategies, and to conduct power system simulations prior to real-world deployment; the keystone of the project.

B. Software

The project deliberately employed straightforward, accessible software tools to demonstrate a realistic cybersecurity threat scenario achievable with minimal resources. Visual Studio Code was selected as the integrated development environment (IDE) for scripting due to its versatility in supporting multiple programming languages, efficient file management, and integrated terminal access to PowerShell and Command Prompt (CMD).

To convert Python scripts into standalone executable applications, PyInstaller was utilized. This freely available tool simplifies the packaging process, offering various options such as creating single-file executables and suppressing console windows to enhance stealth execution.

Additionally, obtaining and becoming familiar with the RSCAD software was a critical step. RSCAD, provided freely for this project, is essential for simulating and analyzing power grid operations within RTDS systems, making it a pivotal component in accurately assessing potential vulnerabilities and their implications

IV. Understanding the Approach

The logic bomb script was strategically designed to emulate realistic cyberattack vectors commonly encountered in operational environments. The ideal attack scenario envisioned involves a victim inadvertently downloading and executing the corrupted application, potentially through social engineering or phishing techniques disguised within legitimate software packages, such as the RSCAD software used with RTDS systems. Upon initial execution, the malicious package generates two distinct executables:

- **WatchdogService**: Continuously monitors and initiates the main logic bomb script.
- **Logic Bomb Script**: Triggered exclusively by the WatchdogService or upon system reboot.

A. Watchdog.exe

Initially running as a discreet background process, the WatchdogService performs essential preliminary tasks to maintain persistence and stealth:

1. **Service Registration**: The WatchdogService logs its activity and subsequently registers itself as a legitimate Windows service to ensure automatic execution upon startup.
2. **Stealth Execution**: Upon securing persistence, the service locates and silently executes a disguised RSCAD executable from a concealed directory.
3. **Continuous Monitoring Loop**: Implements an infinite monitoring loop, continuously verifying the active status of the disguised RSCAD executable and automatically restarting it if terminated, thereby ensuring persistent presence.

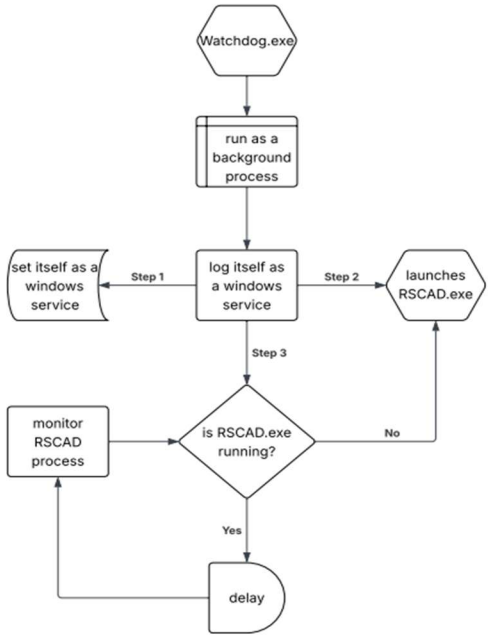

Figure 1. Flow diagram of WatchdogService. This figure represents the flow that watchdog.exe goes through when executed.

B. RSCAD.exe

Upon ensuring persistent execution via the WatchdogService, the logic bomb script undertakes a structured operational procedure comprising three critical stages:

1. **Localhost Proxy Deactivation**: Initially disables any localhost proxy settings by reverting to the default proxy configuration. This preemptive action is essential, as certain systems may attempt port monitoring upon detecting local proxy usage, potentially jeopardizing the script's stealth operation.
2. **Manual Termination Capability**: Implements a continuously running thread allowing manual termination by users. This capability serves dual purposes—providing necessary debugging functionalities and demonstrating multi-threading capabilities within the script.
3. **Port Scanning and Conditional Activation**:
    - Execute an active port scanner continuously monitoring the machine's network connections.
    - Upon detecting an established connection with the RTDS machine, it initiates a controlled delay of one hour before proceeding.
    - Subsequently modifies critical RSCAD simulation files, specifically targeting .dtp files. These files store essential parameters for circuit simulations, such as initial voltage bases, and their alteration causes significant simulation inaccuracies or failures.

While the presence of background processes may eventually draw attention from experienced users, the script leverages multiple layers of obfuscation – first by under legitimate software names and associated official icons to evade initial suspicion. Secondly, by embedding within Windows registry and services, necessitating users to perform complex operations within the Windows Registry Editor and Windows Service Manager for successful removal. These combined measures significantly complicate the detection, analysis, and removal processes, effectively demonstrating vulnerabilities and emphasizing the importance of rigorous cybersecurity practices.

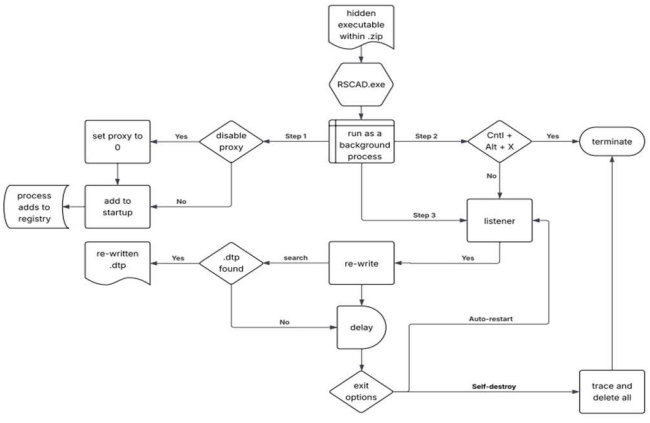

Figure 2. Flow diagram of the main program RSCAD. This figure represents the flow that watchdog.exe goes through when executed.

V. Implemented Libraries

The following Python libraries were meticulously selected and implemented within the project to facilitate specific functionalities essential for the effectiveness and stealthy operation of the project:

1. **winreg**: This library enabled the script to interact directly with the Windows Registry, allowing for creation, modification, and deletion of registry keys. Its primary function was to establish persistence by embedding the script within the Windows startup processes.
2. **os**: Employed extensively for operating system-dependent operations such as creating, deleting, and modifying files and directories. It provided seamless interaction with system-level commands, which were essential for the script's integration with the host environment.
3. **sys**: This library managed Python interpreter interactions, handling execution parameters and environment variables crucial for dynamic script behavior based on runtime conditions and user-defined configurations.
4. **tempfile**: Utilized to securely create temporary directories and files during intermediate execution stages. These temporary elements aided stealth by minimizing traces left in permanent system directories, thereby complicating forensic analyses.
5. **shutil**: Essential for performing comprehensive file operations such as copying, moving, renaming, and deleting files and directories. These functionalities supported both the propagation of the script and subsequent cleanup tasks post-execution.
6. **re (Regular Expressions)**: Implemented for sophisticated pattern-matching tasks within target .dtp files. It precisely identified and modified specific simulation parameters, ensuring that only relevant file components were altered, thus minimizing unintended disruptions.
7. **subprocess**: Facilitated the execution and management of external system-level processes. This capability was crucial for performing stealth executions, launching additional scripts or processes discreetly, and handling interactions with system-level tools.
8. **threading**: Provided robust support for concurrent execution of multiple script tasks, including background monitoring activities, continuous port scanning, and parallel user-interaction handling. Threading ensured the script's responsiveness, reliability, and operational efficiency.
9. **keyboard**: Enabled detection of specific keyboard inputs, crucial for implementing manual termination controls during demonstrations and debugging sessions, thus offering an intuitive and responsive mechanism for script management.
10. **servicemanager**: Allowed seamless integration with the Windows Service Manager, facilitating persistence and ensuring automatic script execution upon system startup. This integration also simplified management

and monitoring of the script's status as a recognized Windows service.

11. **traceback**: This library provided detailed stack trace information, greatly enhancing debugging capabilities. In case of runtime errors, traceback offered precise insights into error sources and script behaviors, facilitating rapid troubleshooting and refinement.
12. **win32event**: Assisted in managing synchronization and communication between multiple threads and processes within the script. This synchronization ensured efficient operation without race conditions, deadlocks, or resource conflicts during multi-threaded execution.
13. **win32serviceutil**: Crucial for creating, installing, managing, and interacting with Windows services. It enabled the WatchdogService to perform automatic service management tasks, significantly enhancing the script's persistence and stealth execution capabilities.
14. **logging**: Implemented comprehensive logging mechanisms to systematically record operational activities, events, and errors. This logging capability provided valuable insights for monitoring script behavior and conducting post-execution analyses.
15. **ctypes**: Facilitated direct interaction with Windows dynamic link libraries (DLLs), thereby extending the script's ability to execute low-level system commands that standard Python libraries could not directly access. It enhanced the script's system-level integration and execution flexibility.
16. **win32com.client**: Enabled interaction with Windows COM (Component Object Model) objects, significantly broadening the range of automation tasks and system-level integrations achievable by the script. This library allowed manipulation and control of various Windows applications and processes, ensuring deep integration with the host system.

Collectively, these libraries constitute a highly integrated and effective toolkit, enabling the development of a robust cybersecurity demonstration. They highlighted the practical applications and significant implications of leveraging Python in cybersecurity contexts, demonstrating the ease of developing sophisticated threat scenarios with readily available tools.

VI. PyInstaller Packaging

A. Script Packaging and Deployment

While packaging might initially appear secondary, its proper execution is critical for delivering the logic bomb effectively and covertly to the targeted host. The goal of packaging in this project was threefold:

1. **Stealth and Integration**: The logic bomb and watchdog scripts were packaged into executable files that run discreetly without generating console windows, thus ensuring stealth during execution.
2. **Icon Mimicry**: To maintain legitimacy and reduce suspicion, it was crucial to use official icons. Specifically, the WatchdogService utilized a standard Windows gear icon, implying a legitimate system service. Conversely, the primary logic bomb executable adopted the official RSCAD icon, significantly enhancing deception by closely mimicking the genuine software.
3. **Single-file Execution**: Both scripts were compiled into single executable files, simplifying delivery and minimizing file-system footprints, further complicating detection.

B. Implementing PyInstaller

To achieve these objectives, the project utilized PyInstaller, a versatile and freely available tool designed for packaging Python scripts into standalone executables. Correct implementation of PyInstaller requires ensuring that Python and PyInstaller paths are properly configured within the system environment variables:

- **Debugging Pathing Issues**: Initially, debugging PyInstaller involves verifying that Python and PyInstaller are accessible through the command line interface (CLI). This involves adding Python's Scripts and bin directories to the system's PATH variable, thus resolving potential issues related to command recognition and execution.
- **Packaging Commands**: Once PyInstaller is properly configured, the packaging commands executed via CMD are straightforward:
    - To package the WatchdogService with the Windows gear icon: *pyinstaller --onefile --noconsole --name WatchdogService --icon winservice.ico watchdog_service.py*
    - To package the logic bomb with the RSCAD icon: *pyinstaller --onefile --noconsole --icon RSCAD.ico --name RSCAD RSCAD.py*

These commands produce compact, stealthy executables that mimic legitimate software, thus significantly enhancing the credibility and effectiveness of the simulated cyberattack scenario.

Fig. 3 Official RSCAD icon

Fig. 4 Official Win. Service Icon

VII. Results

To validate the proof of concept, both the *WatchdogService* and the *logic bomb* executables were deployed together within a controlled environment.

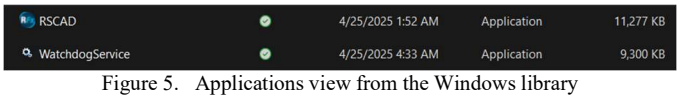

Figure 5. Applications view from the Windows library

A. *Service Initialization and Persistence*

The *WatchdogService* was executed first. Upon launch, it registered itself as a legitimate Windows Service, embedding persistence by configuring automatic startup upon system boot.

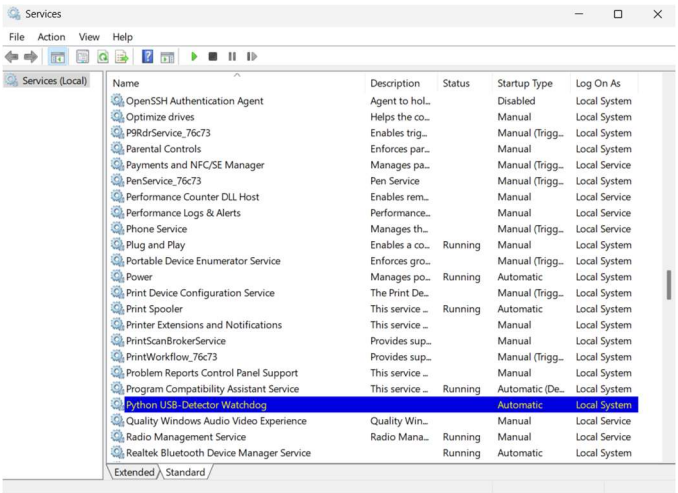

Figure 6. Windows Service registry demonstrating successful implementation

Once active, *WatchdogService* initiated the RSCAD application in stealth mode. This could be verified by observing the RSCAD process running discreetly within the system's background processes.

Figure 7. RSCAD.exe is now embedded as a background process

Additionally, the RSCAD application appeared within the system's *Startup Apps* list, confirming the service's ability to automatically relaunch the application during each system startup.

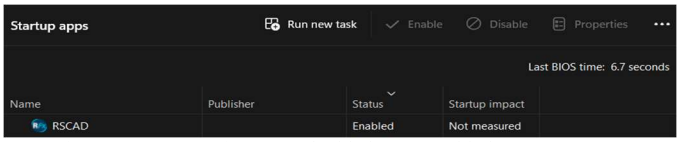

Figure 8. RSCAD is now embedded as an enabled startup process

Further verification using the Windows Registry Editor confirmed the persistence mechanism, revealing registry entries created by the script to ensure re-execution.

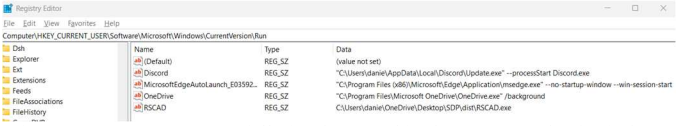

Figure 9. RSCAD successfully implemented in the startup path of Windows registry editor

B. *Logic Bomb Activation and File Manipulation*

Once both the *WatchdogService* and the *logic bomb* were operational, the script actively monitored the specified RSCAD project directory for .dtp files. The logic bomb specifically targeted the *VBASE* parameter, which determines the base voltage of nodes within the smart grid simulation environment.

```
SECTION: NODE_SECTION
   NUMBER_OF_NODES= 39
   INITIAL_NODE_VOLTAGES= 0.0
      NODE= 1 VBASE= 345 IBASE= 1.0 NODENAME= "VBSSNOut"
      NODE= 2 VBASE= 345 IBASE= 1.0 NODENAME= "N10"
      NODE= 3 VBASE= 345 IBASE= 1.0 NODENAME= "N11"
      NODE= 4 VBASE= 345 IBASE= 1.0 NODENAME= "N12"
      NODE= 5 VBASE= 120 IBASE= 1.0 NODENAME= "WFAVAOut"
      NODE= 6 VBASE= 345 IBASE= 1.0 NODENAME= "WFBVAOut"
      NODE= 7 VBASE= 240 IBASE= 1.0 NODENAME= "N25"
      NODE= 8 VBASE= 345 IBASE= 1.0 NODENAME= "N13"
      NODE= 9 VBASE= 120 IBASE= 1.0 NODENAME= "WFAVBOut"
      NODE= 10 VBASE= 345 IBASE= 1.0 NODENAME= "WFBVBOut"
```

Figure 10. Pre-edited .dtp file containing VBASE values

Upon detection of a valid .dtp file, the script altered the *VBASE* value of a randomly selected node, thereby modifying the initial conditions of the simulation.

```
NODE= 5 VBASE= 180 IBASE= 1.0 NODENAME= "WFAVAOut"
```

Figure 11. After execution of logic bomb, demonstrating the altered value of VBASE from 120V to 180V.

C. *Execution Control and Self-Deletion*

After successfully modifying the target file, the script provided two configurable pathways for subsequent behavior:\n\n1. **Repeat Execution**: The script could continue running indefinitely, persistently monitoring and altering .dtp files upon detection.\n\n2. **Self-Destruction**: Alternatively, the script could be configured to auto-delete itself after execution or after a predefined number of activations. This behavior was easily modified by commenting or uncommenting specific lines within the codebase prior to packaging the final executable. This flexible design demonstrated the script's adaptability to various attack strategies, ranging from continuous disruption to stealthy one-time corruption with trace removal.

VIII. Conclusion

This project successfully demonstrated the feasibility and impact of a stealthy, logic bomb-based cyberattack within an RTDS-RSCAD smart grid simulation environment. By leveraging accessible Python libraries and employing standard Windows persistence and obfuscation techniques, the project created a realistic proof-of-concept capable of altering critical simulation parameters without triggering immediate detection. The implementation showcased how a malicious script could maintain persistence through Windows services and registry modifications, execute stealth operations, monitor for specific file conditions, and self-delete to evade forensic analysis. Furthermore, the project highlighted the ease with which relatively simple tools and moderate cybersecurity knowledge could be weaponized to create significant operational disruptions, particularly when security protocols are lax.
Importantly, this work served not as an offensive exploit but as an educational tool to raise cybersecurity awareness among electrical engineering students and professionals. It underscored the pressing need for enhanced security policies, regular audits, and user education in environments where critical infrastructure simulations are performed.

The demonstration validated that even well-defended environments can be vulnerable to simple, overlooked attack vectors, emphasizing the importance of vigilance and proactive defense in safeguarding modern smart grid infrastructures.