RTDS-RSCAD Logic Bomb User Manual

Project Title: RTDS-RSCAD Logic Bomb - A Proof of Concept
Author: Daniele Ricciardelli
Institution: Cal Poly Pomona
Course: ECE 4820-4830 | Cybersecurity SDP
Platform: Windows OS

Table of Contents

1. Introduction
2. Objective & Scope
3. System Requirements
4. Script Architecture
    - 4.1 WatchdogService
    - 4.2 Main Logic Bomb Script (RSCAD.exe)
5. Installation Instructions
6. Execution & Behavior
7. Tips for Demonstration
8. Troubleshooting
9. Future Development Ideas
10. Appendix: File Overview & Library Usage

---

1. Introduction

This manual provides comprehensive instructions and insights into the logic bomb script developed to demonstrate cybersecurity vulnerabilities in RSCAD-based RTDS simulation environments. The intent is educational, showing how system assumptions and user behavior can be exploited.

2. Objective & Scope

The script showcases:

- Stealth persistence as a Windows service
- Conditional logic bomb execution
- File manipulation of .dtp files (e.g., altering VBASE values)
- Optional self-deletion post execution

The purpose is to raise cybersecurity awareness in simulation-driven engineering setups.

3. System Requirements

- OS: Windows 10 or later
- Python 3.x with PyInstaller (for packaging)
- Admin rights (for registry and service install)
- RSCAD software installed or dummy .dtp files

4. Script Architecture

4.1 WatchdogService

Located in watchdog_service.py, this component:

- Registers as a Windows service
- Launches and monitors the main logic bomb
- Relaunches it upon unexpected shutdown
- Logs activity to a local file for traceability

4.2 Main Logic Bomb Script

The RSCAD.py script (packaged as RSCAD.exe) includes:

- Proxy disabling (proxy.py)
- USB-trigger detection (usb_alert.py)
- Manual hotkey-based shutdown (killing.py)
- Registry startup persistence (hidden.py)
- File modification logic targeting .dtp files (action.py)

5. Installation Instructions

1. Ensure Python and PyInstaller are installed and in PATH.
2. Compile the components with PyInstaller:
3. pyinstaller --onefile --noconsole --icon RSCAD.ico --name RSCAD RSCAD.py
4. pyinstaller --onefile --noconsole --icon winservice.ico --name WatchdogService watchdog_service.py
5. Install the WatchdogService as a Windows service:
6. WatchdogService.exe install
7. WatchdogService.exe start
8. Place RSCAD.exe and .dtp files under the configured directory.

6. Execution & Behavior

- On startup, the watchdog launches RSCAD.exe
- RSCAD:
    - Disables proxy settings
    - Registers itself in startup
    - Listens for USB insertions
    - Triggers .dtp file tampering (adds +60 to one random VBASE)
    - Performs optional self-deletion via startup_delete.py

7. Tips for Demonstration

- Modify SEARCH_ROOT in action.py to your environment
- Exclude folders from antivirus scanning to prevent interruptions
- Run services with administrator privileges
- Confirm .dtp files are not open in RSCAD during execution
- Use file hashes (SHA-256) before and after for tamper evidence
- Add print/logging output for live demonstrations

8. Troubleshooting

- **Script doesn't start:** Ensure permissions and paths are correct.
- **No .dtp files found:** Check SEARCH_ROOT in action.py
- **Registry/Service errors:** Run console as administrator
- **Windows Defender blocks execution:** Whitelist script directory

9. Future Development Ideas

- Obfuscation or encryption to avoid AV detection
- Payload delivery via email or macro-enabled docs
- SSH callback or command-and-control server
- Time, user, or behavior-based triggers
- Automatic RSCAD installation on target systems

10. Appendix: File Overview & Library Usage

File Name	Purpose
RSCAD.py	Main script with USB trigger and startup integration
watchdog_service.py	Background monitor and service registry
action.py	File scanning and VBASE modification
hidden.py	Adds script to Windows startup registry
proxy.py	Disables proxy settings
killing.py	Hotkey listener (Ctrl+Alt+X) to kill the script
usb_alert.py	Triggers modification upon USB plug-in
startup_delete.py	Cleanup and self-deletion logic
restart.py	Manual kill, launching logic, and watchdog utilities

Libraries Used:

- winreg, subprocess, threading, os, shutil, tempfile, re, ctypes, keyboard, win32service, win32event, pythoncom, win32com.client