Implementation of a High-Assurance Gateway for Semantic Attack Mitigation in IIoT Modbus TCP Networks

Syed Muhammad Hunain Aghai

October 2026

SUMMARY

This report details the design and implementation of a security gateway prototype designed to protect Industrial IoT (IIoT) infrastructure. The research focuses on the "Semantic Gap" vulnerability, where attackers bypass security gateways by exploiting inconsistencies between network-level policies and device-level logic. A custom Python-based gateway was developed to perform Deep Packet Inspection (DPI) on Modbus TCP traffic. Experimental results demonstrate that the gateway successfully identifies and blocks unauthorized write attempts to safety-critical registers, even when internal state-bypass techniques are employed. The project concludes that high-assurance IIoT security requires semantic-aware protocol enforcement rather than simple packet forwarding.

Contents

1. INTRODUCTION

The rapid integration of Operational Technology (OT) with cloud environments has exposed legacy industrial protocols, such as Modbus TCP, to sophisticated cyber-physical attacks. The objective of this report is to demonstrate a mitigation strategy for "Semantic Gap" attacks. This project implements a High-Assurance Gateway that enforces security invariants at the protocol level. The approach is inspired by contemporary research in formal verification and runtime monitoring, specifically targeting cross-layer inconsistencies in industrial control systems.

2. SYSTEM ARCHITECTURE

The experimental setup consists of a three-zone network architecture:

- **2.1 Field Zone:** A stateful simulator (sim_slave.py) acting as a Programmable Logic Controller (PLC) with internal interlocking logic.
- **2.2 Gateway Zone:** The policy enforcement point (gateway.py) that performs Deep Packet Inspection.
- **2.3 Cloud/Adversary Zone:** An external network from which both administrative commands and malicious injection attacks (attack_payload.py) originate.

3. THE SEMANTIC GAP VULNERABILITY

A semantic gap occurs when a gateway's security model does not perfectly match the PLC’s internal logic. In this study, the gateway was initially configured to monitor only primary authorization registers. However, the attacker targets a "hidden" secondary register (Register 21) to unlock the system state. Because the gateway lacks "semantic awareness" of Register 21, it allows the command through, enabling the attacker to subsequently target safety-critical registers.

4. HIGH-ASSURANCE GATEWAY IMPLEMENTATION

To mitigate the identified vulnerability, the gateway was hardened using Deep Packet Inspection (DPI). The gateway does not merely forward packets; it deconstructs the Modbus TCP Protocol Data Unit (PDU) to identify the Function Code and the Target Register Address in the hex stream.

The security policy is defined as an immutable invariant: Any "Write" attempt (Function Code 06) to the Safety Valve (Register 50 / Hex 0032) must be dropped.

```
# Security Logic Example
if "00060032" in data_hex:
print("[!!! CRITICAL BLOCK !!!] SECURITY VIOLATION.")
continue # Packet drop
```

5. EXPERIMENTAL RESULTS AND DISCUSSION

The system was tested against three phases of attack. The results are summarized in Table 5.1 below.

Table 5.1: Performance of Gateway Under Different Security Configurations

Test Phase	**Attack Vector**	**Gateway Logic**	**Outcome**
Phase 1	Direct Write	Blind Forwarding	System Compromised
Phase 2	State Bypass	Partial Monitoring	System Compromised
Phase 3	Policy Injection	Deep Packet Inspection	Attack Blocked

As shown in Figure 5.1 (see Appendix for screenshots), the Phase 3 gateway successfully identified the malicious hex signature and prevented the unauthorized value (666) from reaching the simulator.

6. CONCLUSIONS

The project successfully demonstrates that standard network-layer security is insufficient for IIoT environments. By implementing a semantic-aware gateway, we can enforce safety invariants that are resilient to state-bypass attacks. Future work should focus on automating policy generation using formal methods to ensure that gateway configurations stay synchronized with evolving PLC logic.

7. REFERENCES

1. Akon, M., Yang, T., Dong, Y. and Hussain, S. R. (2023). "Formal Analysis of Access Control Mechanism of 5G Core Network." *CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 666-680. Available at: https://doi.org/10.1145/3576915.3623113.
2. Hussain, S. R., Mehnaz, S., Nirjon, S. and Bertino, E. (2018). "Secure Seamless Bluetooth Low Energy Connection Migration for Unmodified IoT Devices." *IEEE Transactions on Mobile Computing*, 17(4), pp. 927-944. Available at: https://doi.org/10.1109/TMC.2017.2739742.
3. Hussain, S. R., Nirjon, S. and Bertino, E. (2019). "Securing the Insecure Link of Internet-of-Things Using Next-Generation Smart Gateways." *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 101-108. Available at: https://doi.org/10.1109/DCOSS.2019.00032.
4. Modbus Organization (2024). *Modbus Messaging on TCP/IP Implementation Guide*. [Online]. Available at: http://www.modbus.org.
5. Pymodbus Project (2025). *Pymodbus Documentation v3.x*. [Online]. Available at: https://pymodbus.readthedocs.io/.
6. Wang, W., Cicala, F., Hussain, S. R., Bertino, E. and Li, N. (2020). "Analyzing the attack landscape of Zigbee-enabled IoT systems and reinstating users' privacy." *WiSec '20: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 133-143. Available at: https://doi.org/10.1145/3395351.3399349.

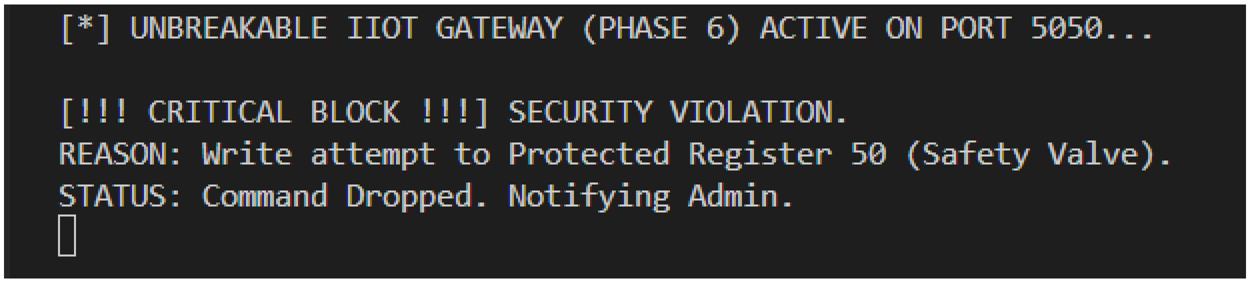
8. APPENDICES

Appendix A: System Logic Code

```
import socket

import threading

GATEWAY_PORT = 5050

SIMULATOR_IP = "127.0.0.1"

SIMULATOR_PORT = 5020

def bridge_traffic(client_socket):

    sim_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sim_socket.connect((SIMULATOR_IP, SIMULATOR_PORT))

    while True:

        data = client_socket.recv(1024)

        if not data: break

        data_hex = data.hex()

        # --- THE UNBREAKABLE POLICY ---

        # We define Register 50 (0032) as a "Critical Asset"

        # that can NEVER be modified through this gateway.

        if "00060032" in data_hex: # Function Code 06 (Write) + Register 50
(0032)

            print("\n[!!! CRITICAL BLOCK !!!] SECURITY VIOLATION.")

            print("REASON: Write attempt to Protected Register 50 (Safety
Valve).")
```

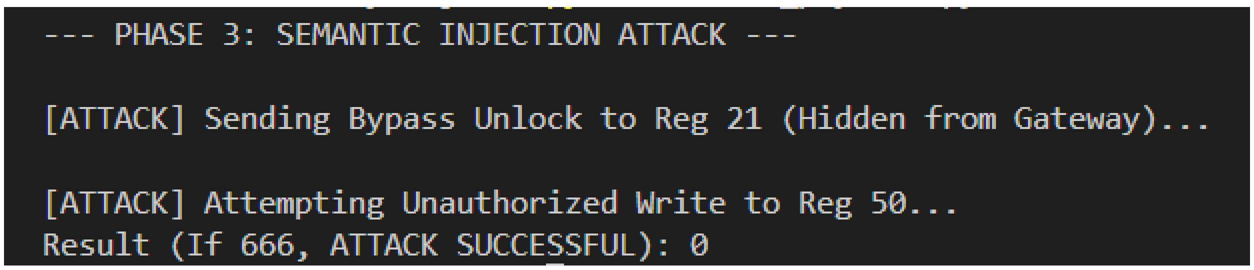
```
            print("STATUS: Command Dropped. Notifying Admin.")

            # Send Modbus Error back to attacker

            error_response = bytes.fromhex(data_hex[:14] + "8601")

            client_socket.sendall(error_response)

            continue # DO NOT FORWARD TO SIMULATOR

        # Forward all other "Safe" traffic

        sim_socket.sendall(data)

        response = sim_socket.recv(1024)

        client_socket.sendall(response)

    sim_socket.close()

    client_socket.close()

if __name__ == "__main__":

    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server.bind(("0.0.0.0", GATEWAY_PORT))

    server.listen(5)

    print(f"[*] UNBREAKABLE IIOT GATEWAY (PHASE 6) ACTIVE ON PORT
{GATEWAY_PORT}...")

    while True:

        client_sock, addr = server.accept()

        threading.Thread(target=bridge_traffic, args=(client_sock,)).start()
```

Appendix B: Experimental Verification



```
[*] UNBREAKABLE IIOT GATEWAY (PHASE 6) ACTIVE ON PORT 5050...

[!!! CRITICAL BLOCK !!!] SECURITY VIOLATION.
REASON: Write attempt to Protected Register 50 (Safety Valve).
STATUS: Command Dropped. Notifying Admin.
```

Figure B.1
Terminal output of the High-Assurance Gateway blocking a malicious write attempt



```
--- PHASE 3: SEMANTIC INJECTION ATTACK ---

[ATTACK] Sending Bypass Unlock to Reg 21 (Hidden from Gateway)...

[ATTACK] Attempting Unauthorized Write to Reg 50...
Result (If 666, ATTACK SUCCESSFUL): 0
```

Figure B.2
Simulator logs showing system state integrity maintained during the attack.