MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Reconnaissance of digital substation protocols

Bachelor's Thesis

Author
Donát Treszler

Advisor
dr. Tamás Holczer

December 4, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Treszler Donát*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. december 4.

Treszler Donát
hallgató

Kivonat

Ez a dolgozat az alállomás-automatizálás fejlődését vizsgálja, kiemelve ezek kritikus szerepét a modern villamosenergia-rendszerekben. Az alállomások az elektromos hálózat kulcselemei, amelyek feszültségátalakítást, rendszervédelmet és energiaáramlás-irányítást végeznek. A hagyományos alállomások manuális működésen és elektromechanikai eszközökön alapultak, amelyek lassúak és hibára hajlamosak voltak. Az *Alállomás-automatizálási Rendszerek (SAS)* és az *IEC 61850* szabvány bevezetése forradalmasította az alállomásokat, lehetővé téve a hatékony kommunikációt, a kábelezés csökkentését és a valós idejű megfigyelést, valamint hibafelismerést.

A fejlesztések ellenére a digitális automatizálás új kiberbiztonsági kihívásokat hozott. Az alállomások sebezhetők a kiberfenyegetésekkel szemben, mint például az üzenetmódosítás és a szolgáltatásmegtagadásos támadások. Ez a dolgozat egy Python-alapú eszközt fejlesztett ki, amely elemzi és teszteli az olyan kulcsfontosságú kommunikációs protokollok biztonságát, mint a *Generic Object Oriented Substation Event (GOOSE)* és a *Sampled Values (SV)*. Az eszköz csomagszűrést, adatfeldolgozást és alapvető támadások szimulációját kínálja, segítve a sebezhetőségek azonosítását és a hálózati viselkedés megértését.

Az eszközt valósághű környezetben teszteltem, és értékes betekintést nyújtott az alállomási kommunikációs rendszerekbe, kiemelve a biztonság és a megbízhatóság javításának lehetőségeit. Ez a munka hozzájárul a digitális alállomások biztonságának és teljesítményének fejlesztéséhez egyre összekapcsoltabb világunkban.

Abstract

This thesis explores advancements in substation automation, focusing on their critical role in modern power systems. Substations are key components of the electrical grid, responsible for transforming voltage, protecting systems, and controlling power flow. Traditionally, substations used manual operations and electromechanical devices, which were slow and error-prone. The introduction of *Substation Automation Systems (SAS)* and the *IEC 61850* standard has transformed substations by enabling efficient communication, reducing wiring, and improving real-time monitoring and fault detection.

Despite these improvements, digital automation introduces cybersecurity challenges. Substations are vulnerable to cyber threats like message tampering and denial of service attacks. This thesis addresses these risks by developing a Python-based tool to analyze and test the security of key communication protocols like *Generic Object Oriented Substation Event (GOOSE)* and *Sampled Values (SV)*. The tool offers packet filtering, data analysis, and simulations of basic attacks, helping identify vulnerabilities and understand network behavior.

Tested in realistic scenarios, the tool provides insights into substation communication systems and highlights ways to improve their security and reliability. This work contributes to advancing the safety, security and performance of digital substations in an increasingly connected world.

Chapter 1

Introduction

1.1 Introduction to Substation Automation Systems (SAS)

Substations are critical components of the electrical grid, serving as key nodes for the distribution and regulation of power. The primary function of substations is to transform voltage levels to facilitate the safe transmission and efficient distribution of electricity across various regions. These transformations are necessary to minimize power losses during transmission and to provide voltage levels suitable for use by residential, commercial, and industrial customers.

In the past, substations were manually operated, requiring human intervention for controlling switches, breakers, and transformers. This manual approach had inherent limitations, including slower response times, higher error rates, and increased downtime. With the growing demand for reliability and efficiency in power grids, substation automation became a necessity. As a result, modern substations have embraced *Substation Automation Systems (SAS)*, which integrate advanced digital technologies to automate critical functions, improve monitoring, and reduce human intervention.

1.1.1 The Evolution of Substations

Historically, substations were constructed with simple electromechanical devices such as relays and manual switches. These devices were slow to respond to faults, and their maintenance required significant manual labor. With the advent of microprocessors and the increasing complexity of power networks, utilities started implementing more advanced systems capable of automating protection, control, and monitoring functions.

Substation automation systems were introduced as part of this technological revolution. By integrating *Intelligent Electronic Devices (IEDs)* and modern communication protocols, substations evolved from manually controlled systems to highly automated, data-driven hubs. These advancements have improved system reliability, reduced operational costs, and increased the speed of fault detection and correction.

1.1.2 Why We Need Substations [11]

Substations play several critical roles in ensuring the efficient and reliable operation of the power grid:

Figure 1.1: Mavir substation[9]

- **Voltage Transformation**: Power is generated at high voltages to minimize transmission losses over long distances. Substations transform these high voltages into lower, more manageable levels for local distribution.
- **System Protection**: Substations house key protective devices, including relays and circuit breakers. These devices detect abnormal conditions, such as short circuits or overloads, and isolate faulty sections of the network to prevent widespread outages.
- **Power Flow Control**: Substations regulate the flow of electricity across different parts of the grid. They help ensure that the supply meets demand and prevent overloads by rerouting power as needed.
- **Real-Time Monitoring and Automation**: With the integration of digital automation systems, substations continuously monitor grid conditions and automatically respond to changes, ensuring the stability and reliability of power delivery.

1.2 The Role of IEC 61850 in Modern Substations

The development of the *IEC 61850* standard marked a significant shift in substation automation. Before its introduction, many substations relied on proprietary communication protocols, which led to challenges in interoperability between devices from different manufacturers. This lack of standardization complicated the integration of new devices, increased operational costs, and limited flexibility.

1.2.1 Overview of IEC 61850

IEC 61850 is an international standard that defines communication protocols for substation automation systems. It was developed with the goal of enabling seamless communication between devices from different vendors while ensuring scalability, flexibility, and interoperability.

At its core, IEC 61850 abstracts the functionality of the substation into *Logical Nodes (LNs)*. Each logical node represents a specific function within the substation, such as voltage measurement, fault detection, or circuit breaker control. These logical nodes are grouped into *Logical Devices (LDs)*, which correspond to larger system components. This abstraction allows for a clear and consistent structure for communication, regardless of the device's hardware or manufacturer.

1.2.2 Key Components of IEC 61850

1. Logical Nodes (LNs): Logical nodes are the fundamental building blocks of IEC 61850. They represent various functions or services within the substation, such as voltage monitoring, current protection, or breaker control. Logical nodes are standardized, ensuring that each node performs a specific function across all devices.

2. Logical Devices (LDs): Logical devices group multiple logical nodes into functional units. For example, a logical device might represent a transformer or circuit breaker that includes several logical nodes responsible for various operations like protection, control, and monitoring.

3. Substation Configuration Language (SCL): One of the most significant features of IEC 61850 is its use of the *Substation Configuration Language (SCL)*. SCL is an XML-based format that describes the configuration of the entire substation. It includes detailed information about devices, communication paths, and system structures. The use of SCL simplifies the setup and maintenance of substations, reducing the potential for configuration errors.

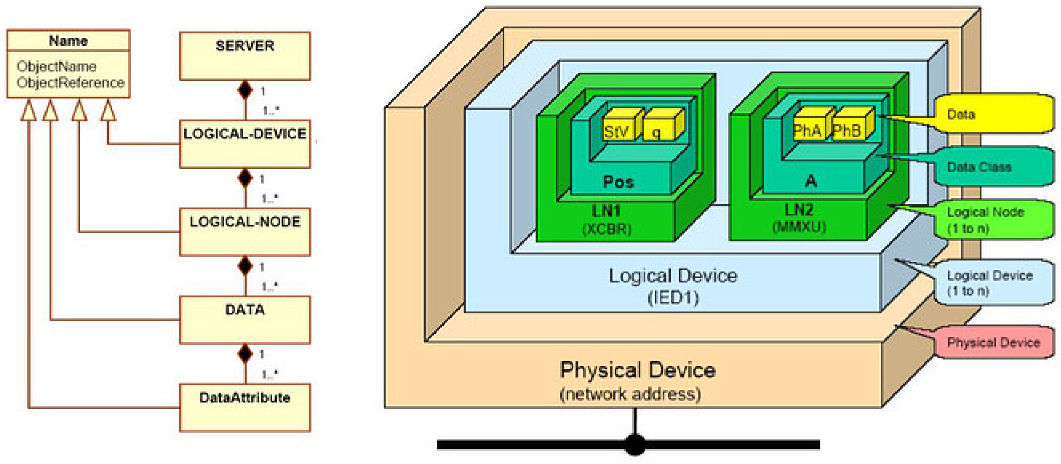


Figure 1.2: Substation Data Model[20]

1.2.3 Advantages of IEC 61850

- **Interoperability**: IEC 61850 ensures that devices from different manufacturers can communicate effectively, regardless of the hardware being used. This eliminates the issue of vendor lock-in and provides utilities with more flexibility in selecting devices.
- **Scalability and Future-Proofing**: The modular design of IEC 61850 allows substations to scale easily. New devices can be added or old ones replaced without requiring significant changes to the overall system.

- **Reduced Wiring and Costs**: Traditional substations relied on extensive wiring to connect individual devices. IEC 61850 reduces the need for physical wiring by utilizing Ethernet-based communication, which significantly lowers installation and maintenance costs.
- **Enhanced System Performance**: IEC 61850 enables real-time data exchange and faster decision-making, leading to improved system performance and reliability. The ability to transmit critical data quickly reduces the time required to detect and isolate faults.

1.3 Substation Network Architecture

A substation's communication network is a sophisticated system that enables efficient data exchange between devices. The network is typically divided into two main buses: the **station bus** and the **process bus**. These buses work together to ensure seamless communication and the smooth operation of the substation.

1.3.1 Station Bus

The **station bus** connects high-level devices within the substation, facilitating the supervision, control, and monitoring of the substation. It serves as the backbone for communication between central control systems and Intelligent Electronic Devices (IEDs), as well as external systems such as SCADA.

Components of the station bus:

- **IEDs (Intelligent Electronic Devices)**: IEDs are crucial elements of the station bus. They perform monitoring, protection, and control functions and serve as intermediaries between the station bus and the process bus.
- **SCADA Systems and HMIs**: Supervisory Control and Data Acquisition (SCADA) systems and Human-Machine Interfaces (HMIs) allow operators to manage and monitor the substation. SCADA systems collect data from IEDs and provide real-time visualization, while HMIs offer localized control capabilities.
- **Communication Protocols**: Protocols such as IEC 61850-8-1 (MMS) enable standardized and reliable communication within the station bus. These protocols ensure efficient data exchange and integration with external systems.

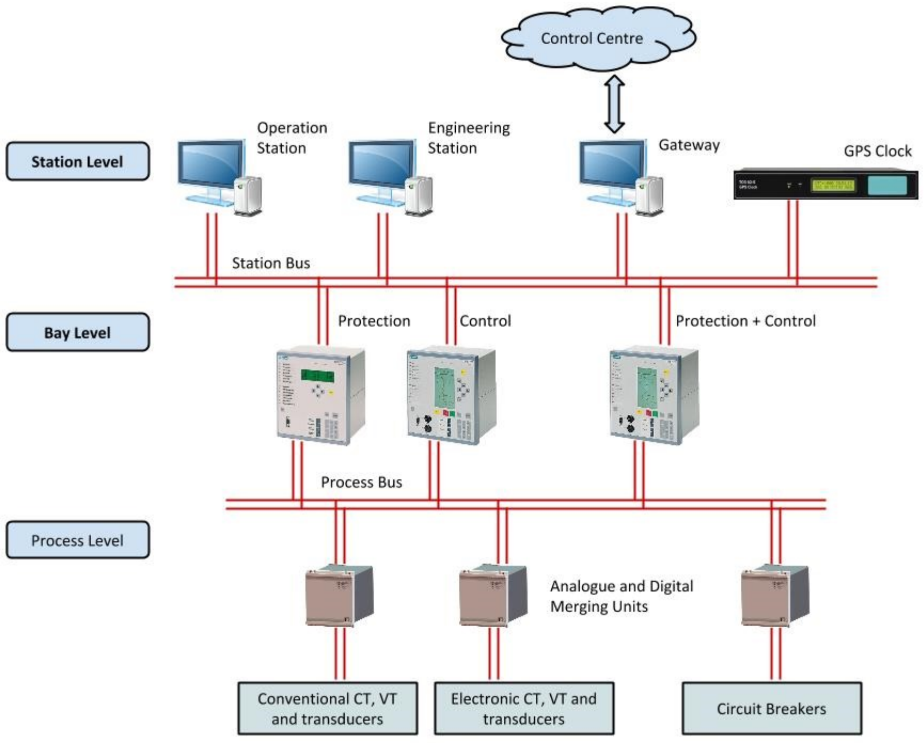


Figure 1.3: Substation Architecture[10]

1.3.2 Process Bus

The **process bus** connects field-level devices, such as sensors and actuators, to IEDs and other components. It replaces traditional copper wiring with digital communication, enabling real-time, high-speed data transfer between protection and control systems and the physical devices in the substation.

Components of the process bus:

- **Merging Units (MUs)**: These devices digitize analog signals from sensors (e.g., current and voltage transformers) and transmit the data to IEDs. Merging units play a critical role in enabling seamless integration of legacy equipment with modern digital systems.
- **Sensors and Actuators**: Sensors measure physical parameters such as voltage, current, and temperature, while actuators (e.g., circuit breakers) execute control commands. These devices communicate directly with the process bus to ensure real-time response.
- **Communication Protocols**: Protocols like IEC 61850-9-2 (Sampled Values and GOOSE messages) provide low-latency and precise communication for real-time protection and control.

1.3.3 Importance of Station and Process Buses

The station bus and process bus are integral to the efficient operation of modern substations. The station bus ensures high-level supervision and coordination between devices, while the process bus facilitates real-time data acquisition and execution of protection and control functions. Together, these buses enhance operational reliability, reduce wiring complexity, and enable seamless integration of advanced technologies within substations.

1.4 GOOSE, SV Protocols, and MSS in Substation Automation [8]

In modern substations, real-time communication is essential for ensuring the safe and reliable operation of the electrical grid. Key communication protocols, such as *GOOSE*, *Sampled Values (SV)*, and the *Manufacturing Subsystem (MSS)*, enable seamless data exchange and interoperability between Intelligent Electronic Devices (IEDs) and other substation components.

1.4.1 GOOSE Protocol

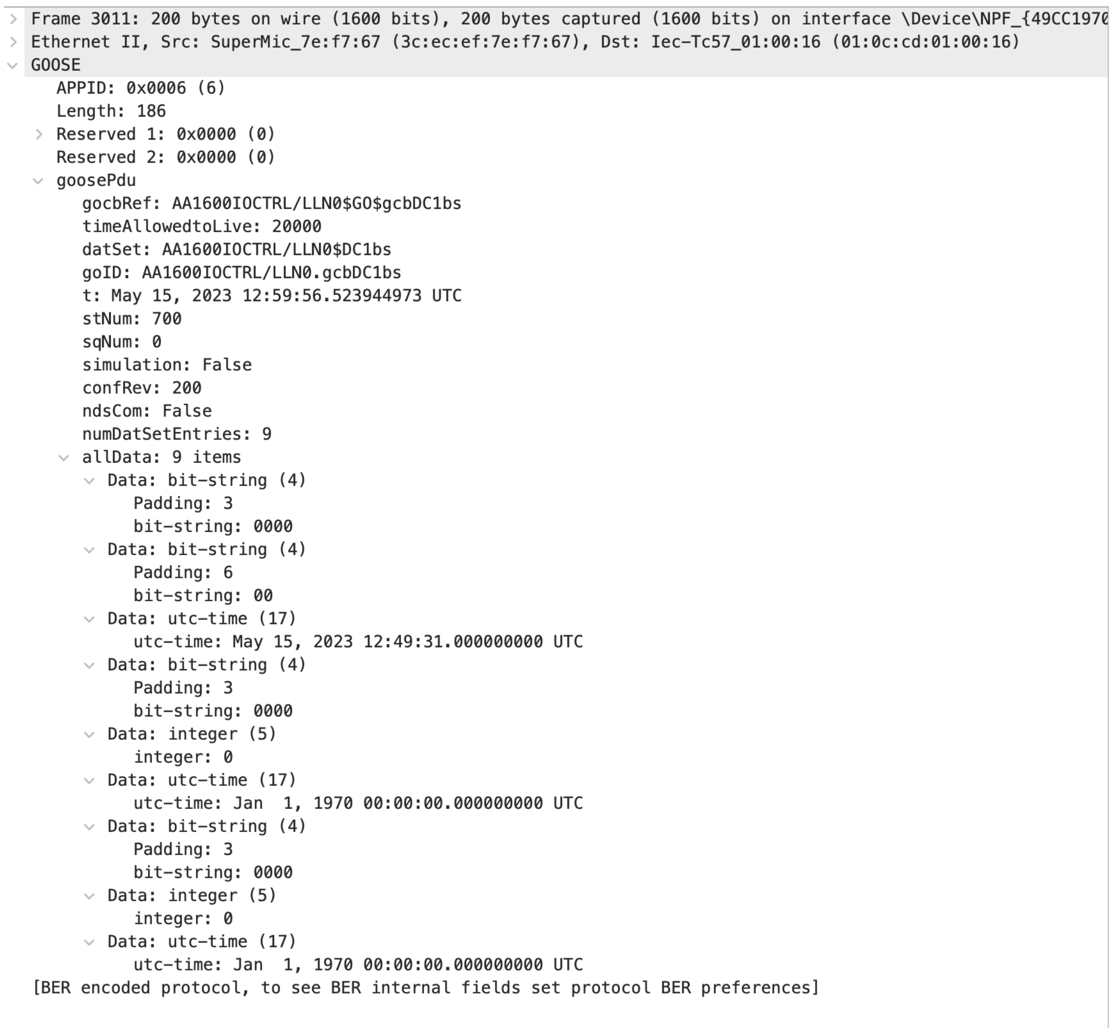
The *GOOSE* (Generic Object-Oriented Substation Event) protocol is an event-driven communication protocol that allows IEDs to exchange real-time information, such as breaker statuses and trip commands. GOOSE is primarily used in protection systems to ensure that faults are detected and isolated quickly.

GOOSE messages are transmitted using *Ethernet multicast*, allowing them to reach multiple devices simultaneously. This ensures rapid communication and minimizes the delay in executing protection actions, such as tripping a circuit breaker in response to a fault.

1.4.2 Sampled Values (SV) Protocol

The *Sampled Values (SV)* protocol is responsible for transmitting real-time measurement data, such as current and voltage, from sensors and merging units to protection and control devices. SV messages are continuously transmitted at high frequencies, ensuring that the control system receives up-to-date information about the substation's operational status.

By providing continuous data streams, the SV protocol enables accurate monitoring of grid conditions and helps prevent system failures by identifying potential issues before they escalate.



```
> Frame 3011: 200 bytes on wire (1600 bits), 200 bytes captured (1600 bits) on interface \Device\NPF_{49CC1970
> Ethernet II, Src: SuperMic_7e:f7:67 (3c:ec:ef:7e:f7:67), Dst: Iec-Tc57_01:00:16 (01:0c:cd:01:00:16)
v GOOSE
    APPID: 0x0006 (6)
    Length: 186
  > Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  v goosePdu
      gocbRef: AA1600IOCTRL/LLN0$GO$gcbDC1bs
      timeAllowedtoLive: 20000
      datSet: AA1600IOCTRL/LLN0$DC1bs
      goID: AA1600IOCTRL/LLN0.gcbDC1bs
      t: May 15, 2023 12:59:56.523944973 UTC
      stNum: 700
      sqNum: 0
      simulation: False
      confRev: 200
      ndsCom: False
      numDatSetEntries: 9
    v allData: 9 items
      v Data: bit-string (4)
          Padding: 3
          bit-string: 0000
      v Data: bit-string (4)
          Padding: 6
          bit-string: 00
      v Data: utc-time (17)
          utc-time: May 15, 2023 12:49:31.000000000 UTC
      v Data: bit-string (4)
          Padding: 3
          bit-string: 0000
      v Data: integer (5)
          integer: 0
      v Data: utc-time (17)
          utc-time: Jan  1, 1970 00:00:00.000000000 UTC
      v Data: bit-string (4)
          Padding: 3
          bit-string: 0000
      v Data: integer (5)
          integer: 0
      v Data: utc-time (17)
          utc-time: Jan  1, 1970 00:00:00.000000000 UTC
  [BER encoded protocol, to see BER internal fields set protocol BER preferences]
```

Figure 1.4: GOOSE packet

1.4.3 Manufacturing Subsystem (MSS) Protocol

The *Manufacturing Subsystem (MSS)* protocol within the IEC 61850 standard facilitates the exchange of data between substation automation systems and external systems such as SCADA (Supervisory Control and Data Acquisition) and other management platforms. MSS enables integration with higher-level applications, ensuring seamless interoperability across different layers of the power system.

MSS utilizes standardized data models and services to exchange operational data, historical records, and configuration information. Key features of MSS include:

- **Device Configuration and Management:** MSS supports the remote configuration of devices and systems, streamlining the deployment and maintenance of substation assets.
- **Data Acquisition for Analysis:** MSS enables the collection of detailed operational and performance data from substation devices, aiding in advanced analytics, fault diagnosis, and system optimization.
- **SCADA Integration:** MSS provides the interface for real-time data exchange with SCADA systems, enabling operators to monitor and control substations efficiently.

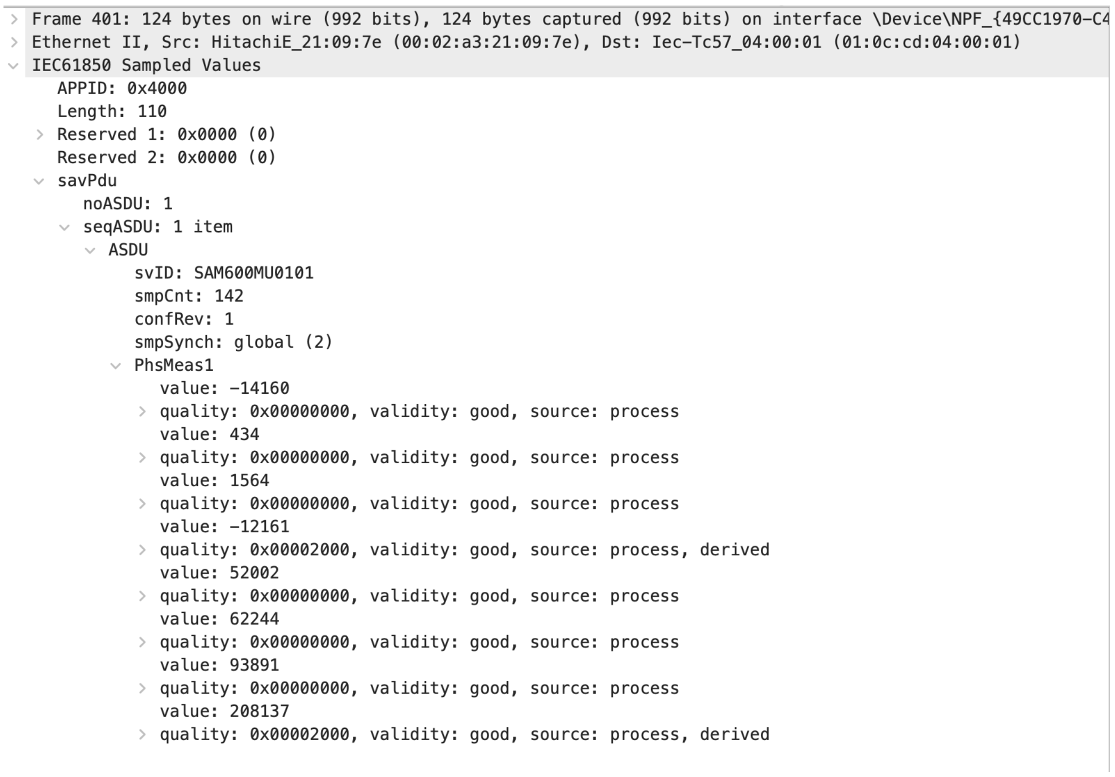

```
> Frame 401: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface \Device\NPF_{49CC1970-C4
> Ethernet II, Src: HitachiE_21:09:7e (00:02:a3:21:09:7e), Dst: Iec-Tc57_04:00:01 (01:0c:cd:04:00:01)
v IEC61850 Sampled Values
    APPID: 0x4000
    Length: 110
  > Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  v savPdu
      noASDU: 1
    v seqASDU: 1 item
      v ASDU
          svID: SAM600MU0101
          smpCnt: 142
          confRev: 1
          smpSynch: global (2)
        v PhsMeas1
            value: -14160
          > quality: 0x00000000, validity: good, source: process
            value: 434
          > quality: 0x00000000, validity: good, source: process
            value: 1564
          > quality: 0x00000000, validity: good, source: process
            value: -12161
          > quality: 0x00002000, validity: good, source: process, derived
            value: 52002
          > quality: 0x00000000, validity: good, source: process
            value: 62244
          > quality: 0x00000000, validity: good, source: process
            value: 93891
          > quality: 0x00000000, validity: good, source: process
            value: 208137
          > quality: 0x00002000, validity: good, source: process, derived
```

Figure 1.5: SV packet

By incorporating MSS, IEC 61850 ensures that substation systems are not only interoperable but also scalable and adaptable to the evolving needs of modern power grids.

1.5 Cybersecurity Challenges in IEC 61850 Networks

While the adoption of digital automation and communication protocols has improved the efficiency of substations, it has also introduced new vulnerabilities. Modern substations are increasingly connected to larger networks, making them susceptible to cyberattacks. Securing IEC 61850 networks is essential to protecting the integrity and reliability of the power grid.

1.5.1 Potential Cybersecurity Threats

Several cybersecurity threats pose risks to IEC 61850-based substations:

- **Replay Attacks**: Attackers can capture legitimate GOOSE or SV messages and retransmit them later, causing devices to perform unintended actions, such as tripping a circuit breaker unnecessarily.
- **Message Modification**: By intercepting and modifying GOOSE or SV messages, an attacker can alter the behavior of protection devices, potentially preventing them from responding to faults.
- **Denial of Service (DoS) Attacks**: Attackers can flood the network with excessive GOOSE or SV messages, overwhelming devices and preventing legitimate messages from being processed.

- **Man-in-the-Middle Attacks**: In this type of attack, an attacker intercepts communication between devices and alters or injects false data, causing incorrect responses and compromising the integrity of the system.

1.5.2 Mitigation Strategies for Securing Substation Networks

To mitigate these cybersecurity risks, utilities must implement a combination of technical and procedural safeguards:

- **Encryption and Authentication**: Implementing encryption protocols and device authentication helps ensure that only authorized devices can communicate with the network and that data remains secure. [18]
- **Intrusion Detection Systems (IDS)**: IDS can monitor network traffic for suspicious activity, allowing operators to detect and respond to potential threats in real-time. [18]
- **Network Segmentation**: Dividing the network into isolated segments can prevent an attacker from gaining control over the entire system if one part of the network is compromised. [18]
- **Regular Firmware Updates**: Keeping devices updated with the latest firmware ensures that security vulnerabilities are addressed and that the system remains protected against known threats. [18]
- **Automated Security Tools:** Advanced security products such as Omicron's *StationGuard* can be deployed to actively monitor substation networks for potential threats and misconfigurations. StationGuard not only detects unauthorized activities and anomalies but also provides detailed insights into operational processes to ensure compliance with IEC 61850 standards. Such tools help utilities achieve proactive and robust network security. [12]
- **Penetration Testing and Vulnerability Assessments:** Tools like Omicron's *CIBANO 500* can support regular testing of network defenses and hardware components. Conducting penetration tests identifies weak points, while vulnerability assessments ensure that substation devices remain resilient against emerging threats. [13]
- **Advanced Monitoring Systems:** Products such as Omicron's *Test Universe* offer comprehensive testing frameworks to validate communication protocols and ensure system integrity. These tools enhance visibility into system performance, enabling operators to preemptively address vulnerabilities. [14]
- **Employee Training and Awareness:** Cybersecurity measures are only effective if personnel are adequately trained to recognize and respond to threats. Utilities should invest in hands-on training programs supported by simulation tools such as Omicron's *RelaySimTest*, which helps staff understand how to maintain security in real-world scenarios. [15]

1.6 Structure of the Thesis

This thesis is organized into several chapters to provide a comprehensive exploration of the substation, its operation, and the solution proposed in this research. The structure is outlined as follows:

Introduction to the Substation and Its Functionality: The first chapter introduces the concept of a substation, describing its key components and operational mechanisms. This section explains the significance of substations in modern infrastructure and their role in facilitating efficient power management and distribution.

Technology Overview: Following the introduction, the thesis delves into the technologies employed in this research. This chapter details the tools, frameworks, and methodologies that underpin the proposed solution, providing an understanding of the technological foundation on which the research is built.

Thesis Goals and Objectives: The next chapter defines the goals of the thesis. It outlines the primary objectives, the challenges addressed, and the research questions that guide this study. This section establishes the scope and the intended contributions of the work.

Theoretical Background: To provide context, the thesis includes a review of relevant literature. This chapter surveys existing research in related areas, highlighting the gaps that this study aims to fill. The discussion includes comparisons to similar works and their methodologies.

Preliminary Work: Building upon the theoretical framework, the thesis presents the preliminary work undertaken during the early stages of this research. This includes exploratory studies, initial experiments, and the foundational insights that informed the subsequent implementation.

Implementation of the Solution: The implementation chapter provides a detailed account of how the proposed solution was realized. It discusses the design, development, and testing processes, supported by technical explanations and architectural diagrams.

Demonstration of the Tool: A practical demonstration of the tool is included to illustrate its functionality. This chapter provides a step-by-step guide on how to use the tool, showcasing its features and usability through real-world scenarios.

Conclusion and Future Work: The final chapter summarizes the contributions of the thesis, reflecting on the outcomes and their implications. It also discusses potential avenues for future research and improvements, emphasizing the broader impact of the findings.

Each chapter is designed to build upon the previous one, ensuring a logical flow of ideas and a clear narrative from introduction to conclusion.

1.7 Summary

The transition from traditional manual substations to fully automated digital substations has revolutionized the power industry. Standards like *IEC 61850* and protocols like *GOOSE* and *Sampled Values (SV)* have enabled real-time communication, improved system reliability, and reduced operational costs. However, with these advancements come new cybersecurity risks that must be addressed to safeguard the integrity of the grid.

By adopting comprehensive security measures, utilities can continue to benefit from the advantages of substation automation while mitigating the risks associated with cyberattacks. As technology evolves, the role of substation automation will become even more critical in ensuring the reliable delivery of electricity to consumers around the world.

Chapter 2

Project Aim and Expected Outcomes

2.1 Introduction

This project is about building a command-line tool called **Substation Sentinel**[1] to analyze network traffic in digital substations, focusing on the GOOSE and Sampled Values (SV) communication protocols. As substations become more digital, real-time monitoring and analysis are essential for smooth operations and cybersecurity. Traditional methods aren't enough anymore due to the increasing complexity, so we need more advanced, automated tools.

Substation Sentinel will offer both real-time and offline traffic analysis, helping engineers and cybersecurity experts in the energy sector. By the project's end, it will be able to inspect packets in detail, show message structures, filter messages by their identifiers, and perform basic analysis of network traffic patterns. Additionally, it will simulate possible attack scenarios to help users understand weaknesses in substation communication protocols.

2.2 Project Aim

The main goal of this project is to create a flexible command-line tool that can analyze both real-time network traffic and previously captured packet data (pcap files). This tool will be a valuable asset for monitoring digital substations, providing detailed analysis of GOOSE and SV messages. Specifically, it will identify, analyze, and display network messages filtered by specific identifiers like Global Object Identifiers (GOID) and Sampled Value Identifiers (SVID).

In addition to its analysis features, the tool will focus on pointing out vulnerabilities in the system, including weaknesses in message structure and communication protocols. The broader aim is to enhance substation security by allowing users to simulate basic attacks, such as replaying messages and modifying values, so they can see how these actions could affect the network's integrity.

[1] `https://github.com/Ralle001/Substation-Sentinel`

2.3 Scope of the Project

This project will develop several key features to achieve our goals. These features are divided into three main areas:

2.3.1 Message Filtering and Display

Our tool will let users filter messages based on their identifiers, like GOID for GOOSE messages or SVID for Sampled Values. This is important for focusing on specific devices or sets of messages in the network traffic. By filtering, network engineers can isolate messages related to certain functions in the substation, such as protection and control systems, ensuring that only relevant traffic is analyzed. This targeted analysis helps in diagnosing issues and checking how the system behaves under both normal and faulty conditions.

2.3.2 Basic Mathematical and Correlation Analysis

A key feature of the tool is its ability to analyze how different devices, GOIDs, and other system parts relate to each other. The goal is to see how changes in one area—like changing a value in a GOOSE message—affect other devices or the whole substation network. By looking at these relationships, users can find out how different components depend on each other, helping them understand how the network is connected.

For example, changing a GOOSE message related to a protection relay might affect the status of a circuit breaker. The tool will track these changes to show how altering one thing affects others. This is especially helpful for finding faults or misconfigurations in the network.

The mathematical analysis will include:

- Finding correlations between how devices behave and specific GOIDs or SVIDs.
- Seeing how changing a value in the network affects other parts, like triggering alarms or changing hardware states.

This analysis helps users get a better understanding of cause-and-effect in the network, providing important information for improving the system and assessing security.

2.3.3 Basic Attack Simulations and Testing in Simulation Environment

Another key part of this project is the ability to simulate basic cyberattacks, which will be built into the tool. The tool will support attacks like:

- **Message Replay Attacks:** The tool will resend previously captured GOOSE or SV messages to simulate replay attacks. This can be used to see how well the system handles repeated or delayed messages.
- **Message Modification Attacks:** In this attack, specific values in a message will be changed before being sent back to the network. This helps to simulate how an attacker could alter critical data, like changing a command to trip a breaker or modifying current or voltage readings.

These attack scenarios will be tested in a simulated environment to observe how the substation network reacts under malicious conditions. The simulation will provide insights into possible weaknesses in protocol implementations and highlight where security measures like authentication or message integrity checks could be improved.

The simulation environment will mimic a real substation network, allowing for realistic testing conditions. This will not only validate the tool's effectiveness but also show the practical risks associated with cyberattacks in digital substation environments.

2.4 Expected Outcomes by End of Semester

The expected outcomes of this project by the end of the semester are as follows:

2.4.1 Functional Command-Line Interface

A fully operational command-line interface (CLI) will be available, capable of accepting user inputs for filtering and displaying pcap files or live traffic streams. The interface will provide intuitive commands for specifying message identifiers such as GOID or SVID, offering users the ability to focus on specific data within the traffic. The CLI will be designed for ease of use, with comprehensive help documentation to guide users through the process of filtering, analyzing, and displaying network traffic.

2.4.2 Comprehensive Message and Traffic Analysis

The tool will provide detailed traffic analysis based on user-defined parameters. This includes the ability to filter traffic based on specific message identifiers and perform basic correlation analysis on the captured data. The results of these analyses will be presented in a clear, readable format, allowing users to quickly interpret the data and make informed decisions regarding the state of the substation network.

2.4.3 Simulation of Cyberattack Scenarios

A key aspect of this project is the simulation of simple cyberattack scenarios. The tool will be able to execute basic cyberattacks, such as message replay attacks and value modification attacks, to highlight vulnerabilities within the GOOSE and SV protocols. These simulations will provide users with a practical understanding of how these attacks work and the potential impact on substation operations. By demonstrating how attackers can exploit protocol weaknesses, the tool will emphasize the importance of implementing proper security measures in digital substations.

2.4.4 Extensibility and Future Development

While our current tool focuses on standard traffic analysis and basic attack simulations, there's plenty of room to expand its capabilities in the future with advanced technologies like Artificial Intelligence (AI) and machine learning. These technologies could enable the tool to automatically detect complex patterns in network traffic and uncover subtle anomalies that might indicate sophisticated cyberattacks or system faults.

AI models could be trained on historical traffic data to spot suspicious behavior or predict potential vulnerabilities in real-time. For example, a machine learning algorithm could detect unusual message patterns, such as unexpected timing or frequency of GOOSE or SV messages, which might suggest an attack or network issue.

However, based on my initial experiments with AI, the results haven't been very promising. Despite its potential, current AI models have struggled to provide useful insights or actionable results in substation traffic analysis. This might be due to factors like the limited availability of labeled training data specific to GOOSE and SV protocols or the complexity of accurately modeling the behavior of digital substations.

In the future, with the right data and advancements in AI technology, AI could play a significant role in enhancing substation security by offering smarter, adaptive methods for traffic analysis and anomaly detection. For now, the project remains focused on more straightforward methods of traffic analysis, with AI considered a promising but currently underdeveloped area for future research.

2.5 Summary

To sum up, this project aims to develop a command-line tool that is a complete solution for analyzing and securing network traffic in digital substations. By offering detailed traffic analysis, real-time monitoring, finding correlations, and simulating attack scenarios, the tool will give valuable insights into how substation communication protocols work and how secure they are. The results of this project will not only add to ongoing research in substation automation security but also provide practical tools for industry professionals working to secure important infrastructure.

Chapter 3

Technological Background

The goal of the **Substation Sentinel** is to analyze these technologies, identify vulnerabilities, and simulate potential attack scenarios. By studying the communication patterns, device configurations, and operational workflows of substations, Substation Sentinel seeks to expose weak points that could be exploited in a cybersecurity context. This analysis provides valuable insights into the risks facing these critical infrastructures.

The following sections delve into the technologies that form the foundation of modern substations, laying the groundwork for their analysis within the Substation Sentinel framework.

3.1 Python

Python is a high-level, interpreted programming language known for its readability and versatility. It is widely used in various domains, including network protocol analysis and cybersecurity, due to its simplicity and extensive library support.

3.1.1 Key Features

- **Easy-to-read Syntax:** Python's clear and concise syntax promotes readability and maintainability, which is crucial for developing complex network analysis tools.
- **Extensive Libraries:** Python offers a vast ecosystem of libraries such as Scapy, PyASN1, and Pandas, which are essential for tasks ranging from packet analysis to data manipulation.
- **Cross-platform Compatibility:** Python is compatible with multiple operating systems, including Windows, macOS, and Linux, facilitating cross-platform development.
- **Rapid Development:** Python's high-level constructs and dynamic typing enable rapid development and prototyping of network tools.

3.1.2 Use Cases

- **Custom Tool Development:** Python can be used to develop bespoke tools for analyzing and manipulating network protocols, such as the tool you are developing based on GooseStalker.

- **Data Analysis and Visualization:** With libraries like Pandas, NumPy, and Matplotlib, Python excels at analyzing and visualizing network traffic data, making it easier to identify patterns and anomalies.
- **Scripting and Automation:** Python scripts can automate routine tasks like packet capturing, data extraction, and report generation, enhancing efficiency and accuracy.
- **Integration with Other Tools:** Python can integrate with other tools and systems, such as databases and web services, enabling comprehensive network analysis solutions.

3.2 Scapy [17]

Scapy is a powerful interactive Python library used for packet crafting, manipulation, and network analysis. It provides a flexible platform for working with network packets and supports a wide range of protocols.

3.2.1 Key Features

- **Packet Crafting and Injection:** Scapy allows users to create and inject custom packets into the network, making it ideal for testing network devices and protocols.
- **Protocol Analysis:** Scapy supports numerous network protocols, enabling detailed analysis and dissection of packet contents.
- **Interactive Interface:** Scapy offers an interactive command-line interface for real-time packet manipulation and analysis.
- **Extensibility:** Users can extend Scapy's functionality by adding support for custom protocols or features.

3.2.2 Use Cases

- **Packet Analysis and Inspection:** Scapy is commonly used for examining and debugging network packets, which is crucial for understanding protocol behavior and vulnerabilities.
- **Security Testing:** Scapy facilitates the development of security tests and exploits by allowing the crafting and sending of malicious packets.
- **Network Testing and Simulation:** Scapy can simulate network traffic to test the resilience of network devices and configurations.
- **Educational Purposes:** Scapy is a valuable tool for learning about network protocols and their vulnerabilities through hands-on experimentation.

3.3 PyASN1 [16]

PyASN1 is a Python library for handling ASN.1 (Abstract Syntax Notation One) encoding and decoding. ASN.1 is used extensively in network protocols to define data structures and formats.

3.3.1 Key Features

- **ASN.1 Parsing and Encoding:** PyASN1 enables the parsing of ASN.1 encoded data into Python objects and vice versa, facilitating the handling of ASN.1-based protocols.
- **Support for Multiple Encoding Rules:** PyASN1 supports various ASN.1 encoding rules, including BER (Basic Encoding Rules), DER (Distinguished Encoding Rules), and PER (Packed Encoding Rules).
- **Modular Design:** The library's modular design allows for flexible handling of different ASN.1 data structures and encoding schemes.

3.3.2 Use Cases

- **Protocol Implementation and Analysis:** PyASN1 is used to implement and analyze protocols that use ASN.1 for data representation, such as telecommunications and security protocols.
- **Data Conversion and Validation:** The library facilitates the conversion of ASN.1 data between different formats and validates data integrity.
- **Custom Protocol Development:** PyASN1 can be employed to develop and test custom protocols that use ASN.1 for encoding.

3.4 Wireshark

Wireshark is a leading network protocol analyzer that captures and inspects network packets in real-time. It provides a comprehensive view of network traffic and is essential for network troubleshooting and security analysis.

3.4.1 Key Features

- **Real-time Packet Capture:** Wireshark captures network packets as they are transmitted, allowing for real-time analysis.
- **Detailed Protocol Decoding:** The tool decodes a wide range of network protocols, providing insights into packet contents and protocol behavior.
- **Advanced Filtering and Search:** Wireshark offers powerful filtering and search capabilities to narrow down and focus on specific packets or traffic patterns.
- **Graphical User Interface:** The intuitive GUI makes it easy to navigate and analyze packet data visually.

3.4.2 Use Cases

- **Network Troubleshooting:** Wireshark helps diagnose network issues by analyzing packet flows and identifying performance bottlenecks.
- **Security Analysis:** The tool is used to detect and investigate security threats and vulnerabilities in network traffic.

- **Protocol Development and Testing:** Wireshark aids in the development and testing of network protocols by providing detailed packet-level information.
- **Educational Tool:** Wireshark is widely used for teaching network protocols and troubleshooting techniques in educational settings.

3.5 GooseStalker [5]

GooseStalker is a specialized tool designed to analyze and exploit vulnerabilities in the GOOSE (Generic Object Oriented Substation Event) protocol, which is part of the IEC 61850 standard used in substation automation systems.

3.5.1 Key Features

- **GOOSE Protocol Analysis:** GooseStalker provides functionality for analyzing GOOSE messages, including inspecting message formats and identifying protocol weaknesses.
- **Exploit Development:** The tool facilitates the development and testing of exploits that target vulnerabilities in GOOSE protocol implementations.
- **Custom Tool Development:** GooseStalker serves as a base for creating customized tools and features for specific needs in substation automation security.

3.5.2 Use Cases

- **Vulnerability Assessment:** Use GooseStalker to assess the security of GOOSE protocol implementations and identify potential vulnerabilities.
- **Tool Customization:** Extend and modify GooseStalker to add new features or adapt it to specific use cases in network security.
- **Security Research:** Conduct in-depth research into the security of substation automation systems and the GOOSE protocol.

3.6 Summary

The combination of Python, Scapy, PyASN1, Wireshark, and tools like GooseStalker provides a powerful toolkit for network protocol analysis and cybersecurity. Each tool contributes unique capabilities that enhance the ability to analyze, test, and secure network communications. By leveraging these tools, researchers and practitioners can develop sophisticated solutions for network analysis, protocol exploitation, and security assessment.

Chapter 4

Theoretical Background

4.1 Overview of Related Works

In this section, we summarize three significant research papers that serve as foundational work for this project. These papers highlight the vulnerabilities and technical challenges in IEC 61850 systems and provide the theoretical underpinnings for the implementation phase.

4.1.1 Research Paper 1: Holistic Attack Methods Against Power Systems [1]

This paper looks at how power systems are using more Information and Communication Technologies (ICTs) and the cybersecurity challenges that come with that. It talks about how older systems, which weren't designed with security in mind, are now vulnerable to cyber threats because they're connected with more advanced, internet-enabled systems.

The paper categorizes different ways attacks can happen, such as physical break-ins, injecting malicious packets, manipulating sequence numbers, and causing denial of service (DoS) attacks. A notable attack is called "ASDU address field starvation," where an attacker fills up the address space, preventing operators from connecting to the device. The research highlights that even with the introduction of the IEC 62351 standard, vulnerabilities still exist in systems based on IEC 61850.

4.1.2 Research Paper 2: Exploiting the GOOSE Protocol [7]

This research examines the vulnerabilities of the GOOSE protocol, specifically how the lack of encryption and authentication makes it easy to exploit. It provides a step-by-step guide on how to attack GOOSE messages by spoofing legitimate messages and sending false information to the Intelligent Electronic Devices (IEDs).

A major takeaway is that the GOOSE protocol is highly susceptible to man-in-the-middle attacks. The research demonstrates how attackers can change key fields (like boolean states) to trigger false actions, such as tripping circuit breakers. This highlights the critical need for security measures in GOOSE communication.

4.1.3 Research Paper 3: IEC 61850 Technology Standards and Cyber-Threats [21]

This paper offers a broad overview of the IEC 61850 standard and the cybersecurity weaknesses of protocols like GOOSE and SV. It critiques the implementation of the IEC 62351 standard, noting that while it provides some security for MMS messages, it fails to secure GOOSE and SV due to the timing requirements of these protocols.

The paper concludes that even though efforts are being made to secure IEC 61850 systems, the most critical protocols remain unprotected, leaving substations vulnerable to various cyber threats.

4.2 Tools for Analyzing and Manipulating GOOSE and SV Messages

Analyzing and changing network traffic in digital substations involves both GOOSE and Sampled Values (SV) messages. These protocols are important for protecting and monitoring substations, but because they lack built-in security, they are vulnerable to cyber-attacks.

One of the main tools is the GooseStalker repository. Initially designed to capture and analyze GOOSE messages, GooseStalker allows users to decode and modify packets. This is especially useful for simulating attack scenarios, like replaying messages or changing them.

However, while GooseStalker was effective for basic GOOSE message analysis, it couldn't handle more complex network setups or analyze SV messages. Since SV messages are crucial for transmitting real-time measurement data, expanding GooseStalker's features to include SV traffic became an important step.

Chapter 5

Preliminary Work

5.1 Extending GooseStalker to Handle SV Packets and Real-Time Traffic

The original GooseStalker tool was missing key features needed for more thorough network analysis in complex substation environments. This project improved GooseStalker in several ways to make it more useful for both GOOSE and SV message analysis.

One of the main additions was support for Sampled Values (SV) messages. By updating GooseStalker to handle SV packets, the tool can now capture, decode, and analyze these messages in both offline (saved pcap files) and real-time situations. This allows users to monitor and manipulate SV messages alongside GOOSE traffic.

The project also focused on enabling real-time traffic analysis. We achieved this by integrating Scapy, a Python library for creating and handling network packets. Scapy's flexibility allowed us to create custom packet layers that could handle more complex setups, including High-availability Seamless Redundancy [6] (HSR) protocols and real-time traffic scenarios.

The enhanced tool now offers several new features:

- **Real-time Packet Capture**: It can capture both GOOSE and SV packets in real-time, allowing for immediate analysis and manipulation.
- **Support for SV Packets**: The tool now supports SV packets, enabling the analysis of continuous data streams that are critical for monitoring substation equipment.
- **Enhanced Packet Manipulation**: With Scapy, users can modify key fields in both GOOSE and SV messages, simulating attacks like replaying or altering messages.
- **Handling Complex Network Layers**: The integration of Scapy also allowed support for more complex network layers like HSR, enabling the tool to handle traffic in advanced substation setups.

These improvements greatly expand what GooseStalker can do, making it a versatile platform for analyzing both GOOSE and SV messages in various substation environments.

5.2 Mathematical Analysis: Correlation Detection and System Impact

A key part of this project involves using mathematical analysis to find connections between different devices and how changes in one part of the system affect others. Substations are complex environments with many devices exchanging data in real-time. Understanding how these data exchanges influence each other is crucial for optimizing performance and preventing problems.

The mathematical analysis focuses on several key areas:

- **Correlation Analysis**: By tracking how GOOSE and SV messages interact with various devices, the tool can detect patterns and correlations. For example, a change in one device's data might trigger changes in others. Identifying these correlations helps operators understand the dependencies within the system.
- **Time-Series Analysis**: The tool measures message frequencies and intervals to detect abnormalities. For instance, unexpected delays in GOOSE message transmission could indicate a malfunction or an ongoing attack.

One specific analysis looks at what happens when a single GOOSE message is altered, like tripping a circuit breaker, and observes how this change spreads through the network. This type of analysis is crucial for diagnosing faults and determining how well the system can handle disruptions.

5.3 Attack Simulations on GOOSE and SV Protocols

As part of the project, we conducted several practical attack simulations using the enhanced GooseStalker tool. These simulations demonstrated the vulnerabilities of GOOSE and SV protocols in real-world scenarios.

5.4 Summary

The theoretical background of this project highlights the critical importance of securing communication protocols in modern substations. As digital substations become more connected and vulnerable to cyberattacks, the need for strong security measures becomes increasingly urgent. By improving tools like GooseStalker and conducting practical attack simulations, this project shows both the weaknesses of the GOOSE and SV protocols and the potential for reducing these risks through better analysis and security measures.

Chapter 6

Implementation

6.1 Packet Differentiation

In network communications, particularly within electrical substations adhering to the IEC 61850 standard, it is crucial to accurately differentiate between various types of network packets. Two fundamental packet types in this context are the Generic Object-Oriented Substation Event (GOOSE) packets and Sampled Values (SV) packets. GOOSE packets are used for rapid, event-driven messaging essential for protection and control systems, while SV packets carry time-critical measurement data like voltage and current samples.

To effectively differentiate between these packet types in our network analysis, we have implemented two Python functions: `goose_test` and `sv_test`. These functions examine the layers and EtherType fields of packets to accurately identify their types.

```
def goose_test(pkt):
    # Check for GOOSE Ether Type in Dot1Q, Ether, or HSR layer
    return any(
        pkt.haslayer(layer) and pkt[layer].type == GOOSE_TYPE
        for layer in ['Dot1Q', 'Ether', 'HSR']
    )
```

Listing 6.1: Function to test for GOOSE packets

The `goose_test` function is designed to identify GOOSE packets by checking if the packet contains any of the following layers: `Dot1Q`, `Ether`, or `HSR` (High-availability Seamless Redundancy). Here's a detailed breakdown of how the function works:

- **Layer Checking**: The function iterates over a list of layers where a GOOSE EtherType might be found. This includes the `Dot1Q` layer for VLAN-tagged packets, the standard `Ether` layer, and the `HSR` layer for networks using seamless redundancy.
- **EtherType Verification**: For each layer present in the packet, the function checks if the `type` field matches the predefined `GOOSE_TYPE` EtherType value (typically `0x88B8`).
- **Result Aggregation**: The `any()` function returns `True` if any of the layer checks pass, indicating that the packet is a GOOSE packet.

This comprehensive approach ensures that GOOSE packets are accurately identified regardless of their encapsulation, which is essential in networks with mixed configurations.

```
def sv_test(pkt):
    # Check for Sampled Values (SV) Ether Type in Ether layer
    return pkt.haslayer('Ether') and pkt['Ether'].type == SV_TYPE
```

Listing 6.2: Function to test for SV packets

The `sv_test` function focuses on identifying SV packets, which typically use the standard Ethernet layer without additional encapsulation. Here's how the function operates:

- **Layer Presence**: It first checks if the packet contains the `Ether` layer.
- **EtherType Verification**: It then verifies if the `type` field in the `Ether` layer matches the predefined `SV_TYPE` EtherType value (commonly `0x88BA`).
- **Result**: The function returns `True` if both conditions are met, confirming the packet is an SV packet.

By specifically targeting the `Ether` layer, the function efficiently identifies SV packets, which are crucial for real-time monitoring and control in power systems.

6.1.1 Understanding the Importance of Layer and EtherType Checks

EtherType values are essential in Ethernet networking as they indicate the protocol encapsulated within the payload of an Ethernet frame. In the context of IEC 61850:

- **GOOSE EtherType (`0x88B8`)**: Used for high-priority, time-critical messaging in protection and control systems.
- **SV EtherType (`0x88BA`)**: Used for transmitting sampled measurement values necessary for real-time analysis and decision-making.

Packets may traverse networks with varying configurations, including VLAN tagging (handled by the `Dot1Q` layer) or redundancy protocols like HSR. Therefore, checking multiple layers ensures that packets are correctly identified regardless of these network variations.

6.2 HSR Layer Implementation in Scapy

High-availability Seamless Redundancy (HSR) is a crucial protocol defined in the IEC 62439-3 standard, designed to provide seamless redundancy in Ethernet networks. It achieves zero recovery time by sending duplicate frames over two redundant paths, ensuring continuous communication even if one path fails. In environments like electrical substations, where network reliability is paramount, HSR plays a vital role.

While Scapy—a powerful Python-based packet manipulation tool—supports a wide range of network protocols, it does not natively support the HSR layer. To analyze and process HSR packets effectively, we implemented the HSR layer within Scapy. This addition allows us to parse HSR packets, access their fields, and integrate HSR handling into our network analysis workflows.

```
from scapy.all import *

class HSR(Packet):
    name = "HSR"
    fields_desc = [
        BitField("path", 0, 4),
        BitField("network_id", 0, 4),
        BitField("lane_id", 0, 4),
        BitField("LSDU_size", 0, 4),
        ShortField("sequence_number", 0),
        ShortField("type", 0),
    ]

bind_layers(Ether, HSR, type=0x892f)
```

Listing 6.3: HSR Layer Implementation in Scapy

6.2.1 Defining the HSR Class

The `HSR` class inherits from Scapy's `Packet` class, allowing us to define the structure of an HSR packet:

- **path (4 bits)**: Identifies the path used for the frame, differentiating between the two redundant paths in an HSR network.
- **network_id (4 bits)**: Specifies the network segment, useful in networks with multiple HSR rings.
- **lane_id (4 bits)**: Indicates the lane or priority level, which can be used for traffic management.
- **LSDU_size (4 bits)**: Represents the size of the Link Service Data Unit, indicating the payload length.
- **sequence_number (16 bits)**: A unique number incremented with each frame, essential for detecting duplicates.
- **type (16 bits)**: Typically the EtherType of the encapsulated protocol (e.g., GOOSE, SV).

By defining these fields, we enable Scapy to parse the HSR header correctly, providing access to each component of the HSR frame for analysis or manipulation.

6.2.2 Binding the HSR Layer to Ethernet

The `bind_layers` function tells Scapy how to interpret packets based on the EtherType:

```
bind_layers(Ether, HSR, type=0x892f)
```

This line binds the `HSR` layer to the Ethernet layer whenever the EtherType is `0x892f`, which is the designated EtherType for HSR. This binding ensures that when Scapy encounters an Ethernet frame with this EtherType, it automatically parses the HSR layer.

6.3 SV Layer Implementation in Scapy

Sampled Values (SV) is a protocol specified in the IEC 61850 standard, used for transmitting time-critical measurement data such as voltage and current samples over Ethernet networks. SV allows for high-speed, deterministic communication essential for real-time monitoring and control in electrical substations.

While Scapy is a powerful tool for packet manipulation, it does not natively support the SV protocol. To analyze and craft SV packets, we implemented the SV layer within Scapy. This custom implementation enables us to parse SV packets, access their fields, and integrate SV handling into our network analysis and testing routines.

```python
from struct import pack

from scapy.packet import Packet
from scapy.fields import XShortField, XByteField, XIntField, ConditionalField, StrLenField
from scapy.all import bind_layers

class SV(Packet):
    name = "SV"
    fields_desc = [
        XShortField("appid", 0),         # Application Identifier
        XShortField("length", 8),        # Length of the SV packet
        XShortField("reserved1", 0),     # Reserved field 1
        XShortField("reserved2", 0),     # Reserved field 2
        XByteField("noASDU", 1),         # Number of ASDUs in the frame
    ]

    def post_build(self, packet, payload):
        sv_pdu_length = len(packet) + len(payload)
        packet = packet[:2] + pack('!H', sv_pdu_length) + packet[4:]
        return packet + payload

class SVPDU(Packet):
    name = "SVPDU"
    fields_desc = [
        XByteField("ID", 0x60),              # Identifier for the SV PDU (Default 0x60)
        XByteField("DefLen", 0x64),          # Length of the PDU
        ConditionalField(XByteField("PDU1ByteLen", 0x00), lambda pkt: pkt.DefLen ^ 0x80 == 1),  # PDU
      1-byte length
        ConditionalField(XShortField("PDU2BytesLen", 0x0000), lambda pkt: pkt.DefLen ^ 0x80 == 2)  #
     PDU 2-byte length
    ]

class ASDU(Packet):
    name = "ASDU"
    fields_desc = [
        XByteField("svID", 0),               # Sampled Value ID
        #XIntField("datSet", 0),              # Data set reference
        XByteField("smpCnt", 0),             # Sample count
        XByteField("confRev", 0),            # Configuration revision
        XByteField("smpSynch", 0),           # Synchronization state
    ]

    def post_build(self, packet, payload):
        # Automatically adjust length of the ASDU if required
        return packet + payload

# Binding SV and SVPDU layers
bind_layers(SV, SVPDU)

# Binding SVPDU and ASDU layers
bind_layers(SVPDU, ASDU)
```

Listing 6.4: SV Layer Implementation in Scapy

6.3.1 Defining the SV Class

The SV class represents the header of an SV packet. It inherits from Scapy's Packet class and defines the following fields:

- **appid (16 bits)**: The Application Identifier distinguishes different applications or data streams using SV.
- **length (16 bits)**: Indicates the total length of the SV packet, including headers and payload.
- **reserved1 (16 bits)**: Reserved for future use, typically set to zero.

- **reserved2 (16 bits)**: Another reserved field, also usually set to zero.
- **noASDU (8 bits)**: Specifies the number of ASDUs (Application Service Data Units) contained in the packet.

The `post_build` method recalculates and updates the `length` field after the packet is built, ensuring it accurately reflects the packet's total size. This adjustment is crucial for proper packet interpretation by receiving devices.

6.3.2 Defining the SVPDU Class

The `SVPDU` class represents the SV Protocol Data Unit, which encapsulates the ASDUs. It includes:

- **ID (8 bits)**: Identifier for the SV PDU, commonly set to `0x60` to indicate an ASN.1 BER-encoded sequence.
- **DefLen (8 bits)**: The definite length of the PDU. If the length exceeds 127, the most significant bit is set, and additional length bytes are used.
- **PDU1ByteLen (8 bits)**: A conditional field for PDU lengths requiring one extra byte.
- **PDU2BytesLen (16 bits)**: A conditional field for PDU lengths requiring two extra bytes.

The use of `ConditionalField` allows the class to handle variable-length encoding based on the value of `DefLen`, accommodating the ASN.1 BER length specifications.

6.3.3 Defining the ASDU Class

The `ASDU` class represents the Application Service Data Unit, carrying the actual sampled values. Its fields are:

- **svID (8 bits)**: The Sampled Value ID, identifying the specific SV stream or data set.
- **smpCnt (8 bits)**: Sample count, incremented with each new sample to track sequence.
- **confRev (8 bits)**: Configuration revision number, indicating changes in the data set configuration.
- **smpSynch (8 bits)**: Synchronization state of the sampled values, important for time alignment.

The `post_build` method in the `ASDU` class can adjust the ASDU's length after building, ensuring consistency and correctness.

6.3.4 Binding the Layers Together

To establish the hierarchical relationship between the layers, we use the `bind_layers` function:

```
# Binding SV and SVPDU layers
bind_layers(SV, SVPDU)

# Binding SVPDU and ASDU layers
bind_layers(SVPDU, ASDU)
```

This binding instructs Scapy to parse the `SVPDU` layer following the `SV` layer and the `ASDU` layer after the `SVPDU` layer. It ensures that when an SV packet is encountered, Scapy correctly interprets the encapsulated layers.

6.3.5 Understanding the Implementation Details

Adjusting Packet Lengths: In network protocols, accurate length fields are essential for proper packet parsing and processing. The `post_build` method in the `SV` class adjusts the `length` field based on the actual packet size:

```
def post_build(self, packet, payload):
    sv_pdu_length = len(packet) + len(payload)
    packet = packet[:2] + pack('!H', sv_pdu_length) + packet[4:]
    return packet + payload
```

This method calculates the total length, packs it into a 16-bit unsigned short in network byte order, and inserts it into the packet at the correct position.

Handling Variable-Length Encoding: The SV protocol uses ASN.1 Basic Encoding Rules (BER) for encoding lengths, which can vary depending on the value:

- For lengths less than 128 bytes, the length is encoded in one byte.
- For lengths 128 bytes or greater, the most significant bit is set to 1, and the remaining bits indicate the number of subsequent bytes used to represent the length.

The `ConditionalField` in the `SVPDU` class handles this encoding by including additional length fields as required.

6.4 Decoding GOOSE and SV Packets Using PyASN1

Generic Object-Oriented Substation Event (GOOSE) and Sampled Values (SV) packets carry critical real-time data encoded in Abstract Syntax Notation One (ASN.1) Basic Encoding Rules (BER) format. To access and interpret this data within our Python-based analysis framework, we utilize the PyASN1 library, which provides tools for decoding ASN.1 BER-encoded data structures.

However, PyASN1 requires ASN.1 specification files to understand the structure of the encoded data. Since the standard ASN.1 specifications for GOOSE and SV packets are not readily available or may need customization, we implemented our own ASN.1 configuration files. These files define the data structures as per the IEC 61850 standard, enabling accurate decoding of the packets.

```python
def goose_pdu_decode(encoded_data):
    # Debugging on
    if DEBUG > 2:
        from pyasn1 import debug
        debug.setLogger(debug.Debug('all'))

    g = IECGoosePdu().subtype(
        implicitTag=tag.Tag(
            tag.tagClassApplication,
            tag.tagFormatConstructed,
            1
        )
    )
    decoded_data, _ = decoder.decode(
        encoded_data,
        asn1Spec=g
    )

    return decoded_data

def sv_pdu_decode(encoded_data):
    # Debugging on
    if DEBUG > 2:
        from pyasn1 import debug
        debug.setLogger(debug.Debug('all'))

    decoded_data, _ = decoder.decode(
        encoded_data,
        asn1Spec=SampledValues()
    )

    return decoded_data
```

Listing 6.5: GOOSE and SV Packet Decoding Functions

6.4.1 Understanding the Decoding Process

PyASN1 Library: PyASN1 is a Python library that provides facilities for ASN.1 data encoding and decoding. It allows us to define ASN.1 data structures in Python and perform BER, DER, CER, or PER encoding/decoding. By leveraging PyASN1, we can parse the ASN.1 BER-encoded payloads of GOOSE and SV packets, extracting meaningful data for analysis.

ASN.1 Specification Files: To decode ASN.1-encoded data, PyASN1 requires the data structures to be defined. We implemented custom ASN.1 specification files for GOOSE and SV packets based on the IEC 61850 standard. These files define the structure and types of the data fields, enabling PyASN1 to interpret the encoded bytes correctly.

6.4.2 Decoding GOOSE Packets

The `goose_pdu_decode` function decodes the ASN.1 BER-encoded payload of a GOOSE packet:

- **Defining the GOOSE PDU**: An instance of `IECGoosePdu` is created with an implicit tag. The tag specifies that the data is an application-specific, constructed type with a tag number of 1. This aligns with the IEC 61850-8-1 standard, which defines GOOSE messages.

- **Decoding the Data**: The `decoder.decode` function parses the encoded data using the provided ASN.1 specification (`asn1Spec=g`). It returns the decoded data and any remaining unparsed data (which should be none if the encoding is correct).
- **Returning the Result**: The function returns the decoded data, which is a PyASN1 object representing the GOOSE PDU with accessible fields.

6.4.3 Decoding SV Packets

Similarly, the `sv_pdu_decode` function decodes the ASN.1 BER-encoded payload of an SV packet:

- **Decoding the Data**: The `decoder.decode` function parses the encoded data using the `SampledValues` ASN.1 specification, which we defined according to the IEC 61850-9-2 standard.
- **Returning the Result**: The decoded data is returned, providing access to the SV packet's fields and values.

6.4.4 Implementing ASN.1 Specifications

GOOSE ASN.1 Specification: [4] The GOOSE messages are defined using ASN.1 in the IEC 61850-8-1 standard. We implemented the `IECGoosePdu` class in Python, mirroring the ASN.1 definition. Key fields include:

- **gocbRef**: GOOSE control block reference.
- **timeAllowedtoLive**: Maximum time the GOOSE message is considered valid.
- **datSet**: Data set reference.
- **goID**: GOOSE message identifier.
- **stNum**: State number, incremented when the dataset changes.
- **sqNum**: Sequence number, incremented with each transmission.
- **test**: Test flag indicating if the message is for testing.
- **confRev**: Configuration revision number.
- **ndsCom**: Needs commissioning flag.
- **numDatSetEntries**: Number of entries in the dataset.
- **allData**: The dataset values.

SV ASN.1 Specification: [19] For SV messages, defined in the IEC 61850-9-2 standard, we implemented the `SampledValues` class. Key fields include:

- **svID**: Sampled Value identifier.
- **smpCnt**: Sample count.

- **confRev**: Configuration revision.
- **smpSynch**: Sample synchronization.
- **data**: The sampled measurement data.

6.4.5 Challenges and Solutions

Lack of Readily Available ASN.1 Specifications: One of the primary challenges was the absence of readily available ASN.1 specification files for GOOSE and SV messages. The IEC standards provide the definitions, but they often require adaptation for practical implementation. To address this, we:

- **Manual Implementation**: Carefully translated the ASN.1 definitions from the standards into PyASN1 classes.
- **Validation**: Tested the implemented classes against known good data to ensure accurate decoding.

Handling Implicit Tags: GOOSE and SV messages use implicit tagging in their ASN.1 definitions. PyASN1 defaults to explicit tagging, so we needed to adjust the tag settings:

- **Setting Implicit Tags**: Used the `subtype` method with `implicitTag` to specify the correct tagging for each class.
- **Understanding Tagging Mechanics**: Ensured that the tags matched the application-specific and constructed nature of the data as per the standards.

Debugging Complex Encodings: ASN.1 BER encoding can be complex, especially with nested structures and optional fields:

- **Using PyASN1 Debugging**: Enabled PyASN1's debugging features to trace the decoding process and identify issues.
- **Incremental Testing**: Decoded individual fields and components separately to isolate problems.

6.5 Decoding IEEE 754 Floating Point Numbers [2]

In the realm of network communications and data analysis, especially when dealing with measurement data like voltage, current, or other sensor readings, it's common to encounter floating-point numbers encoded in hexadecimal format following the IEEE 754 standard. These encoded values need to be converted back into human-readable floating-point numbers for meaningful analysis.

To facilitate this conversion, we implemented a Python function called `floating_point_converter`, which efficiently decodes IEEE 754 floating-point numbers from their hexadecimal representations.

```
def floating_point_converter(hex_string):
    format_type = '<f' if len(hex_string) == 8 else '<d' if len(hex_string) == 16 else None
    if not format_type:
        return 0

    bytes_data = bytes.fromhex(hex_string)[::-1]
    return struct.unpack(format_type, bytes_data)[0]
```

Listing 6.6: Function to Convert IEEE 754 Hexadecimal to Floating Point

6.5.1 Understanding the Conversion Process

The `floating_point_converter` function takes a hexadecimal string as input and returns the corresponding floating-point number. Here's a step-by-step explanation of how the function operates:

Determining the Format Type: The function first checks the length of the input hexadecimal string to determine whether it represents a 32-bit (single-precision) or 64-bit (double-precision) floating-point number:

```
format_type = '<f' if len(hex_string) == 8 else '<d' if len(hex_string) == 16 else None
```

- **Single-Precision (32-bit)**: If the length of `hex_string` is 8 characters (4 bytes), it uses the format specifier `'<f'`, indicating a little-endian 32-bit float.
- **Double-Precision (64-bit)**: If the length is 16 characters (8 bytes), it uses `'<d'`, representing a little-endian 64-bit double.
- **Invalid Length**: If the length doesn't match 8 or 16, the function assigns `None` to `format_type`, and the function returns 0, indicating an invalid input.

Converting Hex String to Bytes: Next, the function converts the hexadecimal string into a bytes object and reverses the byte order:

```
bytes_data = bytes.fromhex(hex_string)[::-1]
```

- **Hex to Bytes**: `bytes.fromhex(hex_string)` converts the hexadecimal string into a bytes object.
- **Byte Order Reversal**: The `[::-1]` slice reverses the bytes. This is necessary because the hexadecimal string may represent data in big-endian format, while the `struct.unpack` function expects little-endian format (as specified by `'<'` in the format string).

Unpacking the Bytes to Float: Finally, the function unpacks the bytes object into a floating-point number:

```
return struct.unpack(format_type, bytes_data)[0]
```

- **Unpacking**: The `struct.unpack` function interprets the bytes according to the specified format and returns a tuple.
- **Retrieving the Value**: The floating-point number is extracted from the tuple using `[0]` and returned.

Challenges in Decoding Floating-Point Data: While testing the proposed methodology, a significant challenge arose when attempting to decode floating-point data, particularly in the context of ABB devices. Unlike standard floating-point representations, ABB appeared to use a non-standard 6-byte format for their floating-point values. This format is longer than the widely-used FLOAT32 (4 bytes) but shorter than FLOAT64 (8 bytes) in the IEEE 754 implementation, presenting unique difficulties in interpretation and conversion.

To mitigate the issue, we attempted several approaches, including truncating or padding the data to conform to 4-byte or 8-byte standards. Additionally, we explored the possibility of the format being a REAL48 representation, a less common floating-point standard, but this too yielded no success. Adjusting the endianness of the data was another strategy we employed, but it did not resolve the issue. Despite these efforts, including countless hours spent researching potential explanations through forums, ABB product documentation, and other online resources, the encoding mechanism remains elusive. This challenge highlights the complexity of working with proprietary or undocumented data formats and underscores the need for clear and standardized data representation in industrial systems.

6.6 Decoding Phasor Measurements [3]

Phasor measurements are vital in power system monitoring and analysis, providing real-time information about voltage and current waveforms. Phasor Measurement Units (PMUs) capture these measurements and often transmit the data in binary format over communication networks. To interpret and utilize this data effectively, it must be decoded from its binary representation into usable numerical values.

We implemented a Python function, `phsmeas`, to decode phasor measurements from a binary string into a tuple of 32-bit integers, extracting the relevant data for further processing.

```python
def phsmeas(binary_string):
    decoded_32bit = struct.unpack(f'>{len(binary_string) // 4}i', binary_string)
    return decoded_32bit[::2]
```

Listing 6.7: Function to Decode Phasor Measurements

6.6.1 Understanding the Decoding Process

The `phsmeas` function operates by unpacking the binary string into 32-bit signed integers and extracting every other integer from the resulting sequence. Here's a detailed explanation of each step:

Unpacking Binary Data: The function uses Python's `struct.unpack` method to convert the binary string into a tuple of integers:

```python
decoded_32bit = struct.unpack(f'>{len(binary_string) // 4}i', binary_string)
```

- **Unpacking Process**:
  - The `struct.unpack` function interprets the binary data according to the specified format string.
  - It returns a tuple of integers representing the unpacked data.

Extracting Relevant Measurements: After unpacking, the function selects every second integer from the tuple:

```
return decoded_32bit[::2]
```

6.7 Attacking Boolean Values in GOOSE Messages

In the context of substation automation and communication, manipulating specific boolean values in GOOSE messages is often necessary for testing and simulation. These messages typically contain data fields that represent the status of different devices, and each boolean value can indicate an operational or fault state. To facilitate control over these values, we implemented a Python function called `toggle_value_by_id`, which allows toggling a boolean value in a GOOSE message based on a specified identifier.

```
def toggle_value_by_id(gd, change_value_id):
    # Helper to toggle a specific boolean by ID
    index = 1
    for item in gd['allData']:
        if hasattr(item, 'getComponentByName'):
            boolean_value = item.getComponentByName('boolean')
            if boolean_value is not None and index == change_value_id:
                item.setComponentByName('boolean', not boolean_value)
            index += 1
```

Listing 6.8: Function to Toggle Boolean Value in GOOSE Message

6.7.1 Understanding the Toggling Process

The `toggle_value_by_id` function iterates through the data structure of a GOOSE message and toggles a boolean value based on the provided identifier. Below is a breakdown of the process:

Identifying the Boolean Component: The function first identifies the boolean component within each item in the message's data structure:

```
boolean_value = item.getComponentByName('boolean')
```

- **Component Retrieval**: The `getComponentByName` method checks if the current data item contains a boolean component.
- **Condition Check**: If a boolean value is found and its identifier matches `change_value_id`, the function proceeds to toggle it.

Toggling the Boolean Value Once the boolean value is located, the function inverts it to simulate a state change:

```
item.setComponentByName('boolean', not boolean_value)
```

- **Inversion**: The boolean value is toggled by setting it to its opposite (`True` to `False`, or vice versa).

6.8 Device Correlation Analysis

To analyze correlations between devices, we implemented a Python function and supporting classes to calculate percentage differences between data points, structure device correlation data, and identify changes over time. This process enables comparison of device measurements, providing insights into whether changes in one device correspond to changes in another.

6.8.1 The Need for Correlation Analysis

Correlation analysis is important because it helps us figure out if devices in the system depend on each other. The goal is to understand how changes in one device's messages or data might affect other devices in the network.

This is useful because if we know which devices are connected or influenced by each other, we can predict the effects of modifying one device's data. This makes it easier when we are testing or attacking the system, as we can focus on the devices that have the biggest impact on others. Instead of randomly changing data, we can plan better by targeting the devices that matter most.

By doing this, we also learn more about how the system works and find weak points where the devices rely too much on each other. This helps us understand how the network might behave when something goes wrong or when we purposely make changes.

6.8.2 Main Correlation Code

The main code iterates through each device in the `dictIDs` dictionary, evaluating correlations between data changes across devices. Instances of `CorrelationClass` are created to store the correlation data for each device, while `CorrelationElement` instances capture specific data change events.

```python
end_list = []
for dev_id in dictIDs:
    device_name = dictIDs[dev_id]
    device_correlation = []
    previous_data = None

    for item in (g for g in gooseData if g.getDatSet() == device_name):
        current_data = item.getAllData()
        is_changed = int(previous_data != current_data)
        device_correlation.append(CorrelationElement(is_changed, item.getTime()))
        previous_data = current_data

    end_list.append(CorrelationClass(device_name, device_correlation))

# Output correlation data for each device
for device in end_list:
    print(device.returnData())

# Compare all pairs of devices for correlation
for (i, device1), (j, device2) in combinations(enumerate(end_list), 2):
    print(f"\nComparing devices {device1.dev_name} and {device2.dev_name}:")

    list1_filtered = device1.filterData()
    list2_filtered = device2.filterData()

    correlation_found = calculate_correlation(list1_filtered, list2_filtered,
     ALLOWED_CORRELATION_DELTA)

    if correlation_found:
        print(f"Correlation found between {device1.dev_name} and {device2.dev_name}.")
    else:
        print(f"No correlation between {device1.dev_name} and {device2.dev_name}.")
```

Listing 6.9: Main Code for Device Correlation Analysis

Explanation of the Process:

- The code iterates through each device, initializing a `CorrelationClass` instance for each device.
- For each device, it loops through the `gooseData` dataset, comparing data entries with the current device.
- For each entry:
  - A `CorrelationElement` with `changed = 1` is created if the data differs from the previous entry, indicating a change.
  - If the data is the same as the previous entry, `changed = 0` signifies no change.
- The correlation data for each device is printed, then pairwise comparisons are conducted to check for correlation between devices.

6.8.3 Percentage Difference Calculation

The `percentage_difference` function calculates the percentage difference between two values, quantifying the change between data points.

```
def percentage_difference(value1, value2):
    if value1 == 0 and value2 == 0:
        return 0.0

    difference = abs(value1 - value2)
    average = (value1 + value2) / 2

    return (difference / average) * 100
```

Listing 6.10: Function to Calculate Percentage Difference

Calculation Process:

- If both values are zero, the function returns 0.0 to avoid division by zero.
- The percentage difference is calculated as the absolute difference divided by the average, multiplied by 100.

6.8.4 Correlation Classes and Data Storage

The correlation data is stored using `CorrelationClass` and `CorrelationElement` classes.

```
class CorrelationClass:
    def __init__(self, dev_name, dev_data):
        self.dev_name = dev_name
        self.dev_data = dev_data

    def returnData(self):
        return " ".join(str(i.changed) for i in self.dev_data)
    def filterData(self):
        return [i for i in self.dev_data if i.changed == 1]

class CorrelationElement:
    def __init__(self, changed, timestamp):
        self.changed = changed
        self.timestamp = timestamp
```

Listing 6.11: Classes for Correlation Data Storage

Class Descriptions:

- `CorrelationClass`:
  - Stores the name and data for each device.
  - Provides methods:
    - `returnData` – Summarizes all changes for the device.
    - `filterData` – Extracts only the changes with `changed = 1`.
- `CorrelationElement`:
  - Stores data change events with a `changed` flag (1 for change, 0 for no change) and a `timestamp` of the event.

6.8.5 Correlation Calculation Between Devices

The `calculate_correlation` function checks for correlation between two lists of `CorrelationElement` instances.

```
def calculate_correlation(list1, list2, allowed_delta):
    count = sum(
        1 for obj1, obj2 in zip(list1, list2)
        if percentage_difference(obj1.timestamp, obj2.timestamp) < allowed_delta
    )
    return count == max(len(list1), len(list2))
```

Listing 6.12: Function to Calculate Correlation Between Devices

Correlation Calculation Process:

- The function iterates over pairs of elements from two lists.
- It counts instances where the percentage difference between timestamps is less than `allowed_delta`.
- A correlation is detected if the count of matching changes meets the length of the longer list, suggesting consistent change timing between devices.

This approach to device correlation analysis enables us to identify interdependencies in device data and reveal potential patterns in data changes.

Chapter 7

Testing

During the course of this thesis, I worked with a variety of network traffic packet capture files (pcaps) to test and refine the functionality of **Substation Sentinel**. These pcaps included datasets that I generated locally for testing purposes, allowing me to evaluate the tool in a controlled environment. All local tests were conducted using simulated data to ensure that the analysis functionality worked as expected.

In addition to my own datasets, I was fortunate to have access to the substation simulator provided by the Faculty of Electrical Engineering and Informatics. This simulator allowed me to test **Substation Sentinel** in a setting that closely mimics real-world conditions. Through their support, I was able to analyze network traffic during two distinct scenarios: normal substation operation and scenarios where the substation was under simulated attack. This real-life-like testing environment was invaluable in validating the effectiveness of my solution and ensuring its ability to identify vulnerabilities in substation communication protocols.

7.1 How to Attack GOOSE Packets

In this section, we will explore how to use a Python-based tool to analyze and attack GOOSE (Generic Object-Oriented Substation Event) packets. The process involves capturing, analyzing, and optionally modifying GOOSE packets in real-time to understand and exploit their behavior.

7.1.1 Launching the Tool

The tool, implemented in Python, can be launched from the command line using the following syntax:

```
python3 goose_device_cnt_realtime.py en1 True
```

Here is a breakdown of the arguments:

- `python3`: Specifies the Python interpreter to execute the script.
- `goose_device_cnt_realtime.py`: The name of the Python script to run.
- `en1`: Specifies the network interface to monitor for GOOSE traffic. Replace `en1` with the name of the desired interface.

- `True`: Determines whether to enable attack mode. Use `False` if the attack functionality is not required.

7.1.2 Capturing and Analyzing Traffic

Once the script is launched, it begins listening for GOOSE packets on the specified interface. As the packets are captured, they are analyzed and categorized. To stop the scanning process and view the results, press `CTRL + C`. After stopping the capture, the tool displays:

- A list of devices detected in the substation network.
- An option to view all messages sent by a specific device by entering the corresponding device number.

7.1.3 Attack Modes

The tool offers multiple attack functionalities for GOOSE packets. After selecting a device and message, the following attack options are available:

- **Replay with Incremented Counter:** Resend the captured message with an incremented counter value to mimic a legitimate update.
- **Rename Device ID:** Modify the device identifier in the message while also incrementing the counter value to disguise the change.
- **Boolean Manipulation:** Change Boolean values within the message, increment the counter, and resend the modified packet.

7.1.4 Correlation Analysis

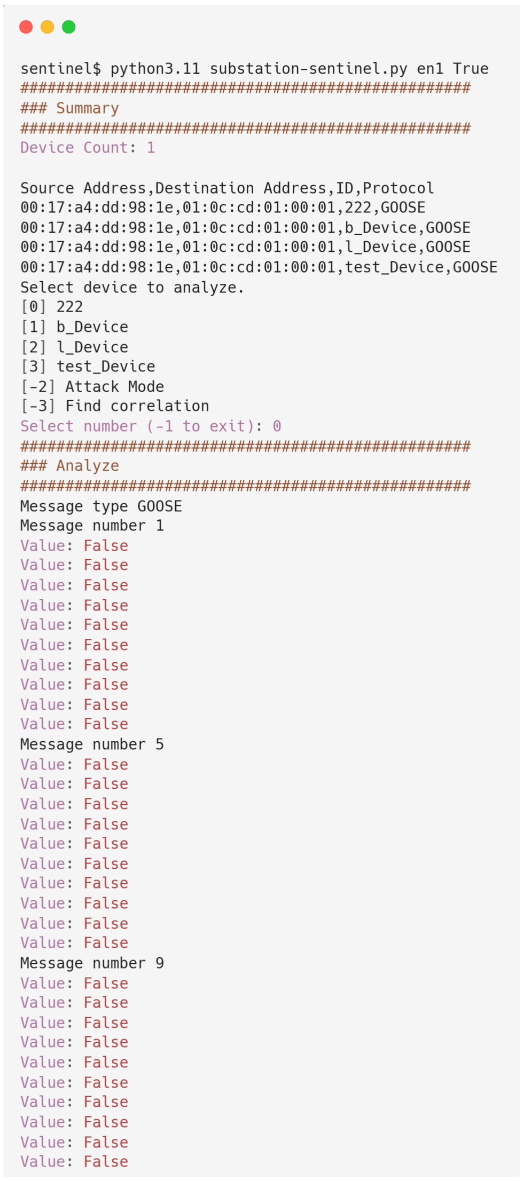
In addition to attacking individual messages, the tool provides a correlation analysis mode. This mode scans the captured packets to identify potential correlations between devices in the network. Such correlations can reveal dependencies or vulnerabilities that may be exploited for further analysis or attacks.

7.1.5 Visualizing the Attack Process

The attack process begins by launching the tool and capturing GOOSE traffic in real-time. Figure 7.1 demonstrates the initial output of the tool, where it identifies the devices on the network and displays their corresponding source and destination addresses, IDs, and protocols.

Once the device analysis is complete, users can enter attack mode by selecting the appropriate option. In attack mode, the tool offers multiple attack types, such as replaying a message, renaming a device, or changing Boolean values in the message.

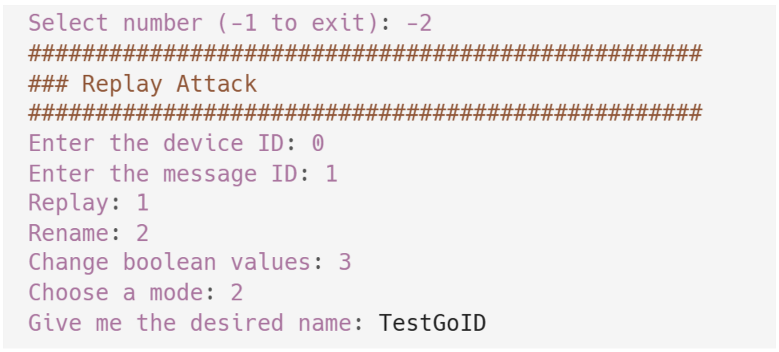
Figure 7.2 illustrates the process of selecting a device and message for a replay attack. In this example, the user chooses to rename the device ID by selecting mode 2 and providing a new identifier.



```
sentinel$ python3.11 substation-sentinel.py en1 True
##################################################
### Summary
##################################################
Device Count: 1

Source Address,Destination Address,ID,Protocol
00:17:a4:dd:98:1e,01:0c:cd:01:00:01,222,GOOSE
00:17:a4:dd:98:1e,01:0c:cd:01:00:01,b_Device,GOOSE
00:17:a4:dd:98:1e,01:0c:cd:01:00:01,l_Device,GOOSE
00:17:a4:dd:98:1e,01:0c:cd:01:00:01,test_Device,GOOSE
Select device to analyze.
[0] 222
[1] b_Device
[2] l_Device
[3] test_Device
[-2] Attack Mode
[-3] Find correlation
Select number (-1 to exit): 0
##################################################
### Analyze
##################################################
Message type GOOSE
Message number 1
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Message number 5
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Message number 9
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
Value: False
```

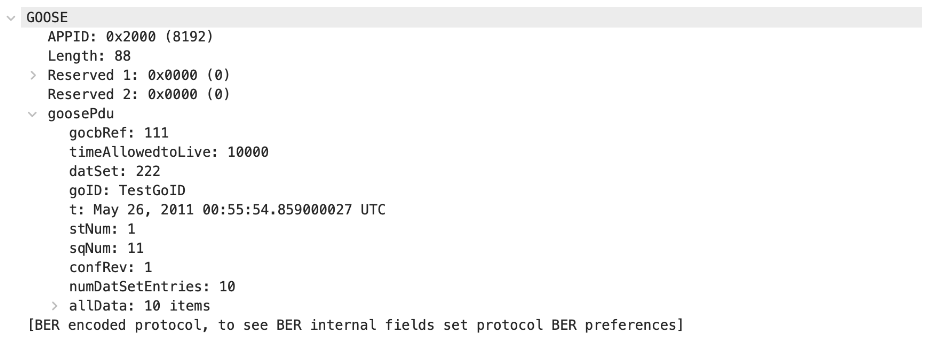
Figure 7.1: Initial output from the GOOSE attack tool, showing detected devices and captured messages.



```
Select number (-1 to exit): -2
##################################################
### Replay Attack
##################################################
Enter the device ID: 0
Enter the message ID: 1
Replay: 1
Rename: 2
Change boolean values: 3
Choose a mode: 2
Give me the desired name: TestGoID
```

Figure 7.2: Executing a replay attack: selecting a device, message, and attack mode (renaming device ID).

After completing the attack setup, the modified packet can be viewed in Wireshark, as shown in Figure 7.3. The replayed packet includes the updated `goID` field (`TestGoID`), demonstrating the successful execution of the replay attack with the new device identifier.



```
GOOSE
  APPID: 0x2000 (8192)
  Length: 88
> Reserved 1: 0x0000 (0)
  Reserved 2: 0x0000 (0)
  goosePdu
    gocbRef: 111
    timeAllowedtoLive: 10000
    datSet: 222
    goID: TestGoID
    t: May 26, 2011 00:55:54.859000027 UTC
    stNum: 1
    sqNum: 11
    confRev: 1
    numDatSetEntries: 10
  > allData: 10 items
[BER encoded protocol, to see BER internal fields set protocol BER preferences]
```

Figure 7.3: Modified GOOSE packet displayed in Wireshark after replay attack.

In addition to attacking specific messages, the tool also includes a correlation analysis feature, which scans the captured packets to identify relationships between devices. This mode helps uncover dependencies or shared characteristics between devices, providing insights into their behavior.

Figure 7.4 demonstrates the correlation analysis process. In this example, the tool compares devices such as 222 and `b_Device`, successfully identifying correlations between their communication patterns. However, it also detects devices that show no correlation, providing valuable network insights.

```
Select number (-1 to exit): -3
1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

Comparing devices 222 and b_Device:
Correlation found between 222 and b_Device.

Comparing devices 222 and l_Device:
Correlation found between 222 and l_Device.

Comparing devices 222 and test_Device:
No correlation between 222 and test_Device.

Comparing devices b_Device and l_Device:
Correlation found between b_Device and l_Device.

Comparing devices b_Device and test_Device:
No correlation between b_Device and test_Device.

Comparing devices l_Device and test_Device:
No correlation between l_Device and test_Device.
```

Figure 7.4: Correlation analysis between devices, showing detected relationships.

This comprehensive process—from capturing packets to performing attacks and analyzing correlations—demonstrates the tool's versatility and effectiveness in both understanding and exploiting GOOSE communication.

7.2 How to Attack SV Messages

This section explores how to use the same Python-based tool to analyze and attack Sampled Value (SV) messages. While the general process of launching the tool and capturing traffic is the same as for GOOSE messages, the attack modes differ slightly due to the nature of SV messages.

7.2.1 Launching the Tool

The process for launching the tool is identical to the one described in the *GOOSE Messages* section. Refer to that section for details on starting the tool, specifying the interface, and enabling or disabling attack mode.

7.2.2 Capturing and Analyzing Traffic

Similar to the process for GOOSE packets, the tool captures SV messages by monitoring the specified interface. While scanning, it identifies both GOOSE and SV messages. Once the desired amount of traffic is captured, pressing `CTRL + C` stops the process and displays the detected devices and their messages.

It is important to note that the tool is not limited to analyzing GOOSE messages—it is equally capable of identifying and working with SV messages. Users can select any detected SV device to further analyze its communication patterns or proceed to attack mode.

7.2.3 Attack Modes

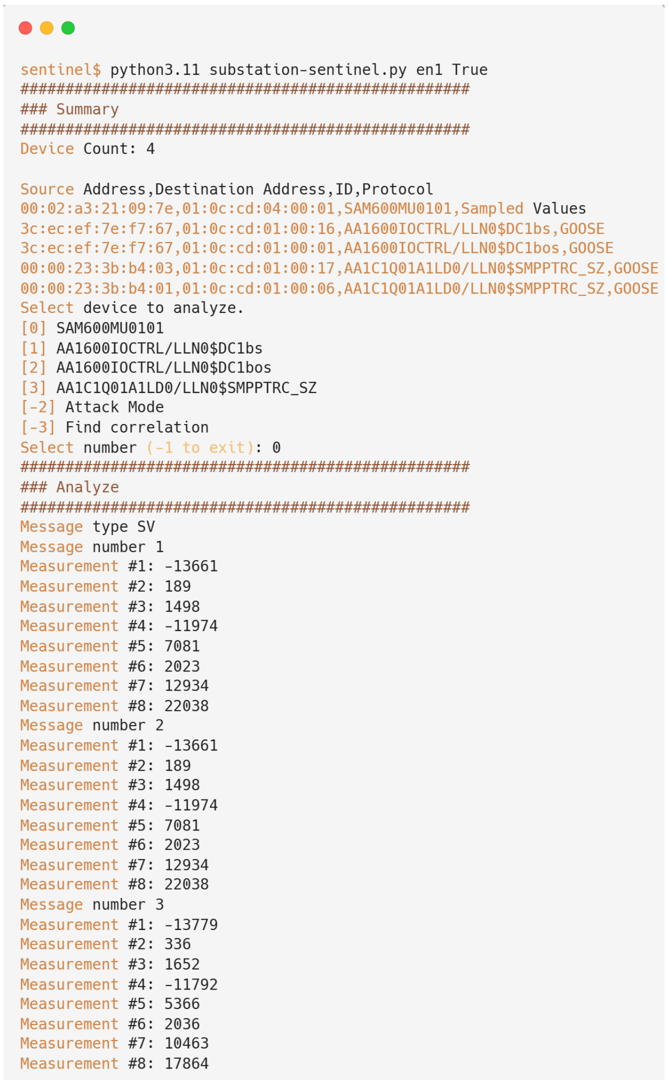
The attack functionality for SV messages shares some similarities with GOOSE attacks but has key differences due to the absence of Boolean values in SV messages. The available attack modes are:

- **Replay Attack:** Resend an SV message with an incremented counter. This mimics legitimate updates to the message, making it difficult to distinguish from normal communication.
- **Rename Mode:** Modify the identifier of an SV device while incrementing the counter. This can mislead devices in the network about the origin of the communication.

Since SV messages do not include Boolean values, the corresponding attack mode available for GOOSE messages does not apply here.

7.2.4 Visualizing the Attack Process for Sampled Values (SV)

The attack process for the Sampled Values (SV) protocol begins by launching the tool and capturing SV traffic in real-time. Figure 7.5 demonstrates the initial output of the tool, where it identifies the devices on the network and displays their corresponding source and destination addresses, IDs, and protocols, along with the sampled measurement values.



```
sentinel$ python3.11 substation-sentinel.py en1 True
##################################################
### Summary
##################################################
Device Count: 4

Source Address,Destination Address,ID,Protocol
00:02:a3:21:09:7e,01:0c:cd:04:00:01,SAM600MU0101,Sampled Values
3c:ec:ef:7e:f7:67,01:0c:cd:01:00:16,AA1600IOCTRL/LLN0$DC1bs,GOOSE
3c:ec:ef:7e:f7:67,01:0c:cd:01:00:01,AA1600IOCTRL/LLN0$DC1bos,GOOSE
00:00:23:3b:b4:03,01:0c:cd:01:00:17,AA1C1Q01A1LD0/LLN0$SMPPTRC_SZ,GOOSE
00:00:23:3b:b4:01,01:0c:cd:01:00:06,AA1C1Q01A1LD0/LLN0$SMPPTRC_SZ,GOOSE
Select device to analyze.
[0] SAM600MU0101
[1] AA1600IOCTRL/LLN0$DC1bs
[2] AA1600IOCTRL/LLN0$DC1bos
[3] AA1C1Q01A1LD0/LLN0$SMPPTRC_SZ
[-2] Attack Mode
[-3] Find correlation
Select number (-1 to exit): 0
##################################################
### Analyze
##################################################
Message type SV
Message number 1
Measurement #1: -13661
Measurement #2: 189
Measurement #3: 1498
Measurement #4: -11974
Measurement #5: 7081
Measurement #6: 2023
Measurement #7: 12934
Measurement #8: 22038
Message number 2
Measurement #1: -13661
Measurement #2: 189
Measurement #3: 1498
Measurement #4: -11974
Measurement #5: 7081
Measurement #6: 2023
Measurement #7: 12934
Measurement #8: 22038
Message number 3
Measurement #1: -13779
Measurement #2: 336
Measurement #3: 1652
Measurement #4: -11792
Measurement #5: 5366
Measurement #6: 2036
Measurement #7: 10463
Measurement #8: 17864
```

Figure 7.5: Initial output from the SV attack tool, showing detected devices and captured messages.

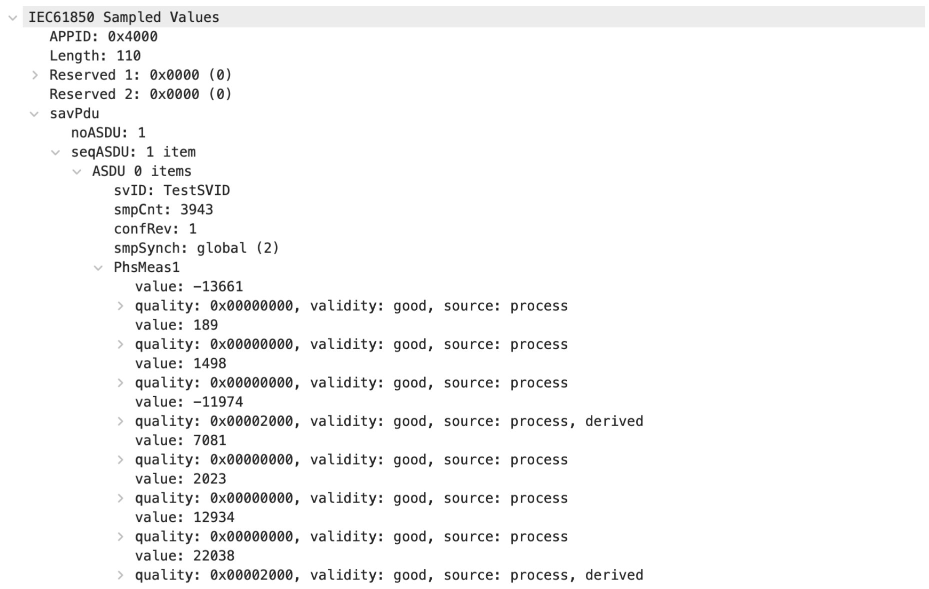
Once the device analysis is complete, users can enter attack mode by selecting the appropriate option. In attack mode, the tool offers multiple attack types, such as replaying a message or renaming a device identifier (e.g., svID).

Figure 7.6 illustrates the process of selecting a device and message for an attack. In this example, the user chooses to rename the svID by selecting mode 2 and providing a new identifier (TestSVID).

```
Select number (-1 to exit): -2
##################################################
### Replay Attack
##################################################
Enter the device ID: 0
Enter the message ID: 1
Replay: 1
Rename: 2
Change boolean values: 3
Choose a mode: 2
Give me the desired name: TestSVID
```

Figure 7.6: Configuring an attack on SV: selecting a device, message, and attack mode (svID renaming).

After completing the attack configuration, the modified packet can be viewed in Wireshark, as shown in Figure 7.7. The updated packet includes the new svID, demonstrating the successful execution of the renaming attack.



```
⌄ IEC61850 Sampled Values
    APPID: 0x4000
    Length: 110
  › Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
  ⌄ savPdu
      noASDU: 1
    ⌄ seqASDU: 1 item
      ⌄ ASDU 0 items
          svID: TestSVID
          smpCnt: 3943
          confRev: 1
          smpSynch: global (2)
        ⌄ PhsMeas1
            value: -13661
          › quality: 0x00000000, validity: good, source: process
            value: 189
          › quality: 0x00000000, validity: good, source: process
            value: 1498
          › quality: 0x00000000, validity: good, source: process
            value: -11974
          › quality: 0x00002000, validity: good, source: process, derived
            value: 7081
          › quality: 0x00000000, validity: good, source: process
            value: 2023
          › quality: 0x00000000, validity: good, source: process
            value: 12934
          › quality: 0x00000000, validity: good, source: process
            value: 22038
          › quality: 0x00002000, validity: good, source: process, derived
```

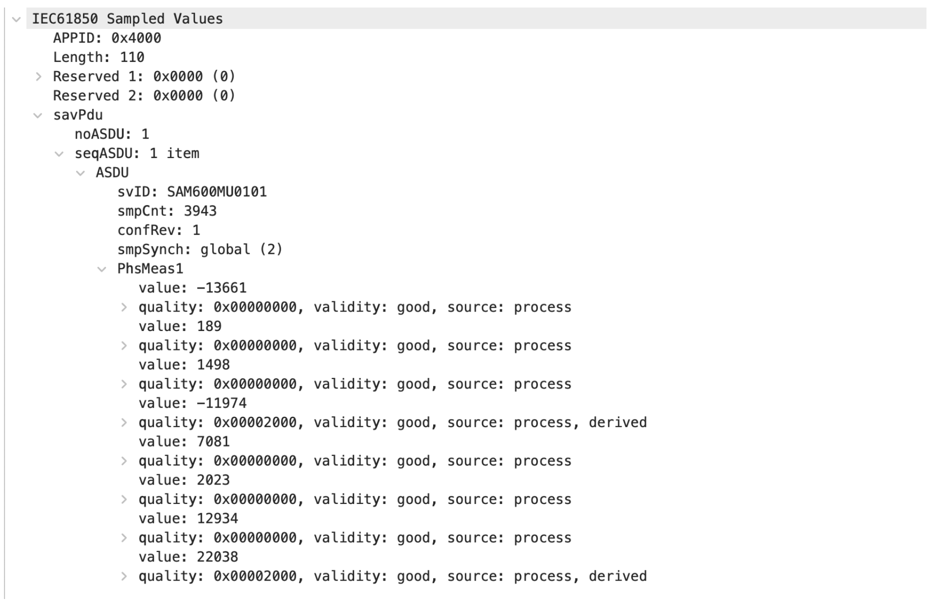
Figure 7.7: Wireshark capture of the SV packet after the renaming attack, showing the updated svID.

In addition to renaming attacks, the tool also enables users to perform replay attacks. Figure 7.8 demonstrates the replay attack process, where the user selects a device and message ID, followed by choosing mode 1 (Replay) to resend the original packet into the network without modification.

The replayed packet can be verified in Wireshark, as shown in Figure 7.9. The captured packet retains its original svID and sampled measurement values under PhsMeas1, demonstrating the successful execution of the attack.

```
Select number (-1 to exit): -2
#####################################################
### Replay Attack
#####################################################
Enter the device ID: 0
Enter the message ID: 2
Replay: 1
Rename: 2
Choose a mode: 1
```

Figure 7.8: Executing a replay attack for SV: selecting device and message for replay.



```
IEC61850 Sampled Values
    APPID: 0x4000
    Length: 110
  > Reserved 1: 0x0000 (0)
    Reserved 2: 0x0000 (0)
    savPdu
        noASDU: 1
        seqASDU: 1 item
            ASDU
                svID: SAM600MU0101
                smpCnt: 3943
                confRev: 1
                smpSynch: global (2)
                PhsMeas1
                    value: -13661
                  > quality: 0x00000000, validity: good, source: process
                    value: 189
                  > quality: 0x00000000, validity: good, source: process
                    value: 1498
                  > quality: 0x00000000, validity: good, source: process
                    value: -11974
                  > quality: 0x00002000, validity: good, source: process, derived
                    value: 7081
                  > quality: 0x00000000, validity: good, source: process
                    value: 2023
                  > quality: 0x00000000, validity: good, source: process
                    value: 12934
                  > quality: 0x00000000, validity: good, source: process
                    value: 22038
                  > quality: 0x00002000, validity: good, source: process, derived
```

Figure 7.9: Wireshark capture of the replayed SV packet, retaining its original structure and values.

This comprehensive process—from capturing packets to performing attacks—highlights the tool's versatility and effectiveness in analyzing and exploiting Sampled Values communication. The ability to modify or replay packets demonstrates the potential risks posed by insecure SV implementations, particularly in critical infrastructure systems where time-sensitive measurements play a pivotal role.

Chapter 8

Future Work

While the current implementation and testing demonstrate the utility and potential of the developed tool, there are several directions for future enhancements and research. These improvements span advancements in Artificial Intelligence, expanded data parsing capabilities, enhanced data manipulation functionalities, and more comprehensive mathematical analyses.

8.1 Integrating Artificial Intelligence for Automated Analysis and Attacks

As Artificial Intelligence (AI) technologies continue to improve, they present significant opportunities to enhance tools like the one developed in this thesis. Future versions of the tool could leverage AI to perform complex tasks such as:

- **Automated Packet Analysis:** AI models could automatically analyze captured network packets, identifying patterns, anomalies, or potential vulnerabilities without manual intervention. These models could quickly adapt to variations in packet structures or protocols, significantly improving the efficiency of network monitoring.
- **Device Correlation and Dependency Detection:** By utilizing advanced machine learning algorithms, the tool could uncover intricate correlations or dependencies between devices that are not immediately apparent through manual correlation analysis. AI could also predict potential cascading effects of faults or attacks within the network.
- **Automated Attack Simulations:** AI could be used to simulate complex and coordinated cyberattacks, optimizing attack strategies to test the resilience of substation networks. This could include discovering new vulnerabilities or identifying unconventional attack vectors.

While AI integration is currently constrained by factors such as data availability and model complexity, future advancements in machine learning algorithms and datasets specific to substation networks could make this a reality.

8.2 Expanding Data Parsing Capabilities

Parsing network data is an ongoing challenge, particularly for proprietary or non-standard formats like the special ABB floating-point representation encountered in this work. Future enhancements could include:

- **Support for Proprietary Formats:** Developing robust parsers for proprietary data formats would enable the tool to handle a broader range of devices and systems. Collaborations with manufacturers or reverse engineering of undocumented formats could aid this effort.
- **Enhanced Decoding for Complex Protocols:** Improved decoding capabilities for nested or dynamic protocol structures would ensure that all critical information is extracted and analyzed effectively.

By expanding its parsing capabilities, the tool could become a more versatile and comprehensive platform for analyzing diverse substation communication protocols.

8.3 Advanced Data Manipulation Features

Currently, the tool supports specific types of data manipulation, such as toggling Boolean values or renaming identifiers. Future versions could enable more granular and versatile data modification capabilities, including:

- **Custom Field Editing:** Allowing users to modify any field within a packet, whether numeric, textual, or binary, would provide unparalleled flexibility in testing and research.
- **Dynamic Payload Alterations:** Tools to create entirely custom payloads or manipulate payloads dynamically based on external inputs or conditions.
- **Scripting and Automation for Modifications:** Introducing scripting capabilities within the tool to automate complex modification scenarios would streamline advanced testing procedures.

These enhancements would allow researchers and engineers to simulate a wider variety of scenarios and test system responses under more diverse conditions.

8.4 Comprehensive Mathematical and Statistical Analysis

Beyond the current correlation analysis, future work could incorporate additional mathematical and statistical techniques to extract more insights from substation data:

- **Advanced Dependency Modeling:** Using techniques such as regression analysis, clustering, or graph theory to model the dependencies and interactions between devices within the network.
- **Statistical Anomaly Detection:** Implementing statistical methods to identify outliers or unusual patterns in network traffic, which could indicate faults, misconfigurations, or potential attacks.

- **Real-Time Metrics and KPIs:** Calculating key performance indicators (KPIs) or real-time metrics that provide actionable insights into the health and efficiency of the substation.

These additional analyses would enable a deeper understanding of substation behavior, enhancing operational efficiency and security.

Chapter 9

Conclusion

This thesis explores the advancements in substation automation, emphasizing the critical role substations play in modern power systems. Substations serve as essential nodes in the electrical grid, facilitating voltage transformation, system protection, power flow control, and real-time monitoring. These functions ensure efficient energy transmission and distribution, supporting the growing demands of residential, commercial, and industrial consumers. Historically, substations relied on manual operations and electromechanical devices, which were slow, error-prone, and required significant human intervention. The advent of *Substation Automation Systems (SAS)* has revolutionized this landscape by integrating digital technologies, reducing human dependency, and improving overall system reliability and performance.

A key milestone in this transformation has been the adoption of the *IEC 61850* standard, which addresses the challenges of interoperability, scalability, and operational efficiency in substation automation. Before IEC 61850, the lack of standardized communication protocols led to difficulties in integrating devices from multiple manufacturers and increased the complexity of substation operations. IEC 61850 resolves these issues by introducing Logical Nodes (LNs), Logical Devices (LDs), and the Substation Configuration Language (SCL), providing a unified framework for seamless communication and efficient configuration of substation components. This standard also reduces physical wiring through Ethernet-based communication, significantly lowering installation and maintenance costs while enabling real-time data exchange for faster fault detection and isolation.

Substation network architecture, comprising the station bus and the process bus, further enhances the operational capabilities of modern substations. The control network connects Intelligent Electronic Devices (IEDs), SCADA systems, and communication protocols, enabling centralized monitoring and remote operation. Meanwhile, the field network integrates physical components such as sensors, actuators, merging units, and circuit breakers, ensuring efficient data collection and device-level control. Together, these networks create a cohesive and robust framework for managing the complex dynamics of power grids.

Communication protocols like *GOOSE* (Generic Object-Oriented Substation Event) and *Sampled Values (SV)* are instrumental in enabling real-time data exchange within substations. GOOSE ensures rapid communication for protection functions, such as fault isolation, while SV provides high-frequency measurement data for accurate monitoring and control. The *Manufacturing Subsystem (MSS)* protocol facilitates integration with higher-level systems like SCADA, enabling efficient device management, data acquisition, and system-wide interoperability.

Despite these technological advancements, the transition to digital automation introduces cybersecurity challenges that must be addressed to ensure the integrity and reliability of the power grid. Substations are increasingly connected to larger networks, making them vulnerable to various cyber threats, including replay attacks, message modification, denial of service (DoS), and man-in-the-middle attacks. To mitigate these risks, utilities must adopt a multi-faceted approach, incorporating encryption, authentication, intrusion detection systems, network segmentation, and regular firmware updates. Advanced tools, such as Omicron's *StationGuard*, can monitor substation networks for potential threats, while simulation-based training programs equip personnel with the skills to maintain robust cybersecurity.

The technological foundation of this thesis is built on a suite of powerful tools that support the analysis, development, and security of network protocols. Python serves as a versatile programming language, enabling rapid development with its extensive libraries, such as Scapy, PyASN1, and Pandas. Scapy, in particular, offers robust capabilities for crafting, inspecting, and injecting network packets, making it invaluable for testing protocol behavior and vulnerabilities. PyASN1's specialization in ASN.1 encoding and decoding facilitates the implementation and analysis of protocols that use structured data representations.

Wireshark, as a real-time network protocol analyzer, complements these tools by providing packet capture, detailed protocol decoding, and advanced traffic analysis. It is indispensable for troubleshooting, educational purposes, and protocol development. GooseStalker extends this ecosystem into substation automation security, targeting vulnerabilities in the GOOSE protocol within IEC 61850 networks. Its ability to analyze GOOSE messages and facilitate exploit testing makes it a key tool for assessing the security of substation communication protocols.

The aim of this project is to develop a command-line tool for analyzing network traffic in digital substations, specifically focusing on GOOSE and Sampled Values (SV) protocols. The tool will provide capabilities for both real-time and offline analysis of network packets, allowing users to filter and inspect messages by identifiers such as GOID and SVID. Beyond analysis, the tool will simulate basic cyberattacks, including message replay and value modification, to highlight vulnerabilities in substation communication protocols.

Key features of the tool include message filtering, basic correlation and mathematical analysis, and attack simulations tested in a realistic substation simulation environment. These capabilities will enable users to diagnose system behavior, identify interdependencies among devices, and assess the impact of potential cyberattacks. The project's expected outcomes include a fully functional command-line interface, detailed message and traffic analysis, and simulated attack scenarios, all of which will contribute valuable insights into the security and functionality of substation communication protocols.

The theoretical background for this thesis establishes the foundational knowledge and literature that underpin the development of advanced tools and methods for analyzing substation communication protocols. It surveys the existing body of research on substation automation, focusing on the vulnerabilities in GOOSE and SV protocols and highlighting areas where current approaches fall short in addressing these challenges.

This section provides critical insights into protocol design, communication standards, and security requirements for digital substations. It also compares existing tools and methodologies, identifying their limitations in real-time monitoring, traffic analysis, and cybersecurity. By addressing these gaps, the theoretical background justifies the need for a flexible and extensible command-line tool, as proposed in this thesis, to advance the state-of-the-art in substation automation security and analysis.

The implementation phase of this thesis focuses on the development of robust methods for analyzing and manipulating GOOSE and SV network packets. Functions like `goose_test` and `sv_test` accurately differentiate these critical packet types, ensuring effective filtering and analysis in IEC 61850 networks. Furthermore, Scapy-based implementations extend support for HSR and SV layers, providing the ability to parse, craft, and interpret specialized network packets, including the integration of ASN.1 decoding using PyASN1 for detailed analysis of GOOSE and SV payloads.

Additionally, functions such as `floating_point_converter` and `phsmeas` decode IEEE 754 floating-point data and phasor measurements, facilitating the interpretation of measurement data. Correlation analysis tools and classes evaluate device interdependencies, while tools for toggling specific values in GOOSE messages enable targeted manipulation for testing and security assessment. Together, these features ensure comprehensive packet analysis, security testing, and enhanced understanding of network dynamics in digital substations.

This implementation supports the broader goals of the project by addressing the complexities of substation communication protocols, providing tools for network engineers and cybersecurity professionals to analyze, test, and secure critical infrastructure.

The testing phase demonstrates the versatility and effectiveness of the Python-based tool developed in this project. For GOOSE packets, the tool enables real-time capture, detailed analysis, and various attack functionalities, such as replaying messages with incremented counters, renaming device IDs, and manipulating Boolean values. Users can also conduct correlation analysis to uncover interdependencies between devices, providing valuable insights into network behavior and potential vulnerabilities.

Similarly, the tool supports the analysis and exploitation of Sampled Values (SV) messages. While sharing some attack functionalities with GOOSE, SV attacks focus on replaying messages and renaming device identifiers, highlighting the risks posed by insecure implementations of time-critical protocols in substation automation.

Through detailed visualization of captured traffic and modified packets in tools like Wireshark, the testing phase validates the practical applications of this tool for both analysis and cybersecurity research. The ability to simulate attacks, analyze correlations, and assess the impact of changes in network communication underscores the tool's utility in securing critical infrastructure against potential threats.

Bibliography

[1] János Csatár, Péter György, and Tamás Holczer. Holistic attack methods against power systems using the iec 60870-5-104 protocol.

[2] Floating point in GOOSE, Last accessed: 2024.11.24. https://osqa-ask.wireshark.org/questions/21169/float-values-in-goose-protocol/.

[3] Guglielmo Frigo and Marco Agustoni. Phasor measurement unit and sampled values: Measurement and implementation challenges. In *2021 IEEE 11th International Workshop on Applied Measurements for Power Systems (AMPS)*, pages 1–6, 2021. DOI: 10.1109/AMPS50177.2021.9586036.

[4] GOOSE protocol, Last accessed: 2024.11.24. https://www.incibe.es/en/incibe-cert/blog/security-goose-protocol.

[5] GooseStalker. https://github.com/cutaway-security/goosestalker.

[6] High-availability Seamless Redundancy, Last accessed: 2024.11.24. https://www.cisco.com/c/en/us/td/docs/solutions/Verticals/Machines/HSR/ConnMach-HSR.pdf.

[7] Juan Hoyos, Mark Dehus, and Timthy X Brown. Exploiting the goose protocol: A practical attack on cyber-infrastructure. In *2012 IEEE Globecom Workshops*, pages 1508–1513, 2012. DOI: 10.1109/GLOCOMW.2012.6477809.

[8] iGrid. https://www.igrid-td.com/smartguide/iec61850/, Last accessed: 2024.11.21.

[9] Magyar Napelem Napkollektor Szövetség. Óriási alállomás-fejlesztési projektet jelentett be a mavir. https://www.mnnsz.hu/oriasi-alallomas-fejlesztesi-projektet-jelentett-be-a-mavir/, Last accessed: 2024.11.21.

[10] Steven McFadyen. Substation architecture. https://myelectrical.com/notes/entryid/245/how-a-digital-substation-works, Last accessed: 2024.11.29.

[11] Aleen Mohammed. https://peguru.com/2012/05/why-d-we-need-electrical-power-substations/, Last accessed: 2024.11.21.

[12] Omicron. https://www.omicronenergy.com/en/products/stationguard/, Last accessed: 2024.11.21..

[13] Omicron. https://www.omicronenergy.com/en/products/cibano-500/, Last accessed: 2024.11.21..

[14] Omicron. https://www.omicronenergy.com/en/products/test-universe/, Last accessed: 2024.11.21..

[15] Omicron. https://www.omicronenergy.com/en/products/relaysimtest/, Last accessed: 2024.11.21..

[16] PyASN.1. https://pyasn1.readthedocs.io/en/latest/contents.html.

[17] Scapy. https://scapy.net/.

[18] Roberto Setola and Francesco Morelli. Cyber security strategies for the protection of electrical substations. https://ceur-ws.org/Vol-3260/paper14.pdf, Last accessed: 2024.11.21.

[19] SV protocol, Last accessed: 2024.11.24. https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/iec_61850_sampled_values_protocol.html.

[20] TestGuy. Exploring iec 61850 substation communication standards overview. https://wiki.testguy.net/t/exploring-iec-61850-substation-communication-standards-overview/2640, Last accessed: 2024.11.21.

[21] Tarek A. Youssef, Mohamad El Hariri, Nicole Bugay, and O. A. Mohammed. Iec 61850: Technology standards and cyber-threats. In *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pages 1–6, 2016. DOI: 10.1109/EEEIC.2016.7555647.