Dissertation on

"Covert Channel Detection and Prevention"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS390A – Capstone Project Phase - 1

Submitted by:

Karan Bhat Sumbly	**PES1UG20CS193**
Kumkum Geervani	**PES1UG20CS184**
Ghanashyam Mahesh Bhat	**PES1UG20CS153**
Prajwal Bhat	**PES1UG20CS290**

Under the guidance of

Dr. Sapna V M
Assistant Professor

January - May 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100 Feet Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Title of the capstone project'

is a bonafide work carried out by

KARAN BHAT SUMBLY	**PES1UG20CS193**
KUMKUM GEERVANI	**PES1UG20CS184**
GHANASHYAM MAHESH BHAT	**PES1UG20CS153**
PRAJWAL BHAT	**PES1UG20CS290**

in partial fulfilment for the completion of sixth semester Capstone Project Phase - 1 (UE19CS390A) in the Program of Study - **Bachelor of Technology in Computer Science and Engineering** under rules and regulations of PES University, Bengaluru during the period Jan. 2023 – May. 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 6th semester academic requirements in respect of project work.

Signature	Signature	Signature	**External Viva**
Dr. Sapna V M	**Dr. Shylaja S S**	**Dr. B K Keshavan**	
Associate Professor	Chairperson	Dean of Faculty	

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the Capstone Project Phase - 1 entitled **"Covert Channel Detection and Prevention"** has been carried out by us under the guidance of Dr. Sapna V M, Associate Professor and submitted in partial fulfilment of the completion of the sixth semester of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2022. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1UG20CS193	**Karan Bhat Sumbly**
PES1UG20CS184	**Kumkum Geervani**
PES1UG20CS153	**Ghanashyam Bhat**
PES1UG20CS290	**Prajwal Bhat**

ACKNOWLEDGEMENT

We would like to express our gratitude to Dr. Sapna V M, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE19CS390A - Capstone Project Phase – 1.

We are grateful to the project coordinator, Dr. Priyanka H., all the panel members & the supporting staff for organizing, managing, and helping the entire process.

We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from her.

We are grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, Dr. B.K. Keshavan, Dean of Faculty, PES University for providing us various opportunities and enlightenment during every step of the way.

Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

ABSTRACT

Unidentified covert channels in a network can seriously compromise security, especially in high-security environments where sensitive information needs to be shielded from unauthorized access. The common techniques used to counter covert channels are monitoring, filtering, and encryption but the effectiveness of these defence mechanisms depends on the network's ability to identify and anticipate the covert channels used by the attacker. Thus, we aim to develop a software tool to detect and prevent the use of covert channels.

The "Covert Channel Detection and Prevention" software will provide a means for detecting and preventing covert channels in computer networks. While a wide variety of covert channels exist, this tool will focus on detecting timing channels. The software will be designed to work with various network protocols and be compatible with a wide range of hardware configurations. Additionally, the software will be tested in a virtual environment using multiple virtual machines (VMs) and data transmission through covert timing channels. The software will also be designed to disrupt covert timing channels after they have been detected.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A covert channel is said to be a secret communication channel between two hosts that are utilized to obfuscate data and get beyond security precautions.

The fleeting headway in computing power provides new opportunities in expanding the use of covert channel communications for malware exfiltration of data, orchestrating nodes of a botnet, or sending remote commands [1] while avoiding getting detected by firewalls, intrusion detection systems, and anti-viruses [2]. Although covert channels can be used for achieving sound objectives like implementing digital watermarking in VoIP traffic, developing traceback techniques, or avoiding censorship they are being used for criminal activities making them a threat to cybersecurity [3]. Covert channels pose a serious risk to the privacy and security of the systems since they go mostly undetected because of the low generalizability and scalability of the covert channel detection and analysis tools [1]. The issue arises from the fact that there are indefinite ways to conceal information in a covert channel and the tools developed can only address certain types of covert channels. Thus, we aim to develop a software tool to detect and prevent the use of covert channels.

The "Covert Channel Detection and Prevention" software will provide a means for detecting and preventing covert channels in computer networks. While a wide variety of covert channels exist, this tool will focus on detecting timing channels as they are difficult to detect without a proper mechanism in place. The software will be designed to work with various network protocols and be compatible with a wide range of hardware configurations. We aim to test the software in a virtual environment using multiple virtual machines (VMs) and data transmission through covert timing channels. The software will also be designed to disrupt covert timing channels after they have been detected.

2. PROBLEM STATEMENT

We aim to develop a 'Covert Channel Detection and Prevention' tool that will try to detect and disrupt covert timing channels in computer networks. The tool will be designed to work with a variety of network protocols and hardware configurations. The tool will be tested in a virtual environment using multiple virtual machines and data transmission via covert timing channels.

3. LITERATURE SURVEY

Table 1. Papers surveyed and their Takeaways

PAPER SURVEYED	Takeaway
[1] "Code Augmentation for Detecting Covert Channels: Targeting the IPv6 Flow Label"	This paper aims to make use of the code augmentation features like eBPF to find hidden channels that target the IPv6 Flow Label. This method talks about analysing packets containing flow labels by modifying just the network stack code. It does not require any modifications to the network infrastructure. But this method has a disadvantage as it can increase the performance overhead and processing time.
[2] "Bccstego: A Framework for Investigating Network Covert Channels"	This paper describes a method for investigating Network covert channels using Bccstego which is an inspection framework to identify hidden channels that target network headers. Though the method described in this paper proves to provide high throughput, deep packet inspection poses scalability issues. This method neglects the state of network connection between endpoints, hence aids in less memory usage and less overhead. There are difficulties pertaining to handling v4/v6 conversion. Integration of this tool with other frameworks can increase the performance.
[3]"CCgen: Injecting Covert Channels into Network Traffic"	This paper talks about CCgen, a tool to inject covert channels into the network. CCgen allows for creation of customized covert channels according to specific network configurations and security systems. We can

	inject multiple covert channels in the same capture. Covert channels generated by this method are difficult to be detected using both signature-based and anomaly-based detection techniques. Though this method has the advantages mentioned earlier, it can be used by attackers for carrying out malicious activities.
[4] "PcapStego: A tool for Generating Traffic Traces for Experimenting with Network Covert Channels"	This paper talks about PcapStego, a tool for the generation of network covert channels within the .pcap files. There are two main uses of the tool:1) generation of large real-world traces 2) creates "replayable" conversations helpful for conducting pen testing campaigns or simulating attacks. This tool also helps to generate data for IPv6 channels with varying degrees of complexity and levels of cover channel embedding.
[5] "DAT detectors: uncovering TCP/IP covert channels by descriptive analysis"	This paper introduces us to detectors based on Descriptive Analytics of Traffic to make it easier to expose covert network and transport layer channels that were created using a variety of data-hiding techniques. DAT detectors are known to be fast and does lightweight analysis, hence they are embedded in the Network Intrusion Detection Systems. They tend to be comprehensive and flexible. The first level security covert channel detection is a unique feature of these detectors. Analysis of network traffic is simpler with the help of these detectors because of the usage of multimodality distributions and symbols by which we can easily identify patterns which are

	consistent with covert channels. Challenge regarding the analysis arises when there is noisy and bouncing channels. This method proved to provide no false negatives.
[6] “Packet Length Covert Channel: A Detection Scheme”	This paper talks about Packet Length Covert channels which creates a covert traffic which is like the regular traffic making the detection of such channels extremely difficult. This method can detect covert channels with high accuracy. This paper has mentioned an accuracy of 98% with 0.02 false positive rate. The method mentioned above uses packet length as a carrier hence it cannot be used for other methods which use different carriers. Packet Length Covert Channel does not require a shared key to be exchanged between two communicating entities. This reduces overheads and is more secure. It increases complexity and performance overhead.
[7] "Covert Communication using Address Resolution Protocol Broadcast Request Messages”	This paper introduces us a technique for network covert communication that makes use of broadcast request signals from the Address Resolution Protocol (ARP) to communicate invisibly over a local area network. The suggested method encrypts the secret data using a common seed value that is already known to both the sender and receiver, increasing the secret message's imperceptibility. This method uses random numbers and ASCII code to encode and strengthen the imperceptibility of secret data instead of hiding secret data directly in the

	Target Protocol Address field. Different encoding values are used for the same occurrences of any character which makes frequency analysis difficult. This assumes that the covert sender generates and sends ARP Broadcast Request messages only for covert communications. The method has relatively high bandwidth and low latency, but low reliability and limited message size.
[8] "Covert Channel Detection: Machine Learning Approaches"	This survey paper has described various types of Covert channels. The paper includes a discussion of Covert channel exploitation by IoT devices. The paper introduces us to various Machine learning approaches which aid in the detection of covert channels. We have come across preventive mechanisms for the same along with the challenges faces in the detection of the covert channels. The paper introduces us to various tools and models for the detection. The paper also mentions that SVM (Support Vector Machine) is the best model for the detection of covert channels.
[9] "Code Layering for the Detection of Network Covert Channels in Agentless Systems"	This paper talks about the method which makes use of the eBPF Filter to build ad hoc security layers within virtualized architectures without the requirement for additional agents to be embedded. A rich set of eBPF programs are run and then a condensed statistics on the header fields are collected. It uses a bin-based data structure to reveal hidden communications. This method is efficient and easy to implement. It does not require any

	special agent or hardware for the detection. But it requires access to the source code of the application. Adding additional code can increase overhead.
[10] "ARPNetSteg: Network Steganography Using Address Resolution"	This paper introduces us to an algorithm that mimics Network Steganography using ARP. Sender encodes the covert message by converting the message to a hexadecimal string. This method can transfer 44 bits of covert data per ARP reply packet. The last hexadecimal is used to store the Control information. A seed value is used for random generation of unallocated IP addresses. Both the sender and receiver make use of this seed value. The receiver sends ARP request broadcast message with the generated IP address. This method provides no mechanism for authenticating the source address of a machine sending an ARP response. But the accuracy is increases largely with just 1 retry. Further increase in the retries may not increase accuracy in significant levels.
[11] "Decision Tree Rule Induction for Detecting Covert Timing Channels in TCP/IP traffic"	This paper implements and checks the use of DAT detectors for the network timing channels. DAT detectors consist of three well-defined phases: pre-processing, feature extraction and transformation and detection. To replicate the primary timing strategies described in the literature, a testbed has been developed. Eight covert timing channels have been implemented for the conducted experiments based on packet inter-arrival

	times. The generalised model had an accuracy of 99.18%, precision of 90.95% and recall of 95.95%. This method may need significant computational resources to process large amounts of network traffic and generate decision trees.
[12] "Network Protocol Covert Channels: Countermeasure Techniques"	This paper discusses the detection and prevention of covert channels and introduces the concept of the network covert channel triangle (DSM). In a network covert channel, Network protocol is used as a carrier which is known as a structured carrier. The paper talks about such various detection methods. These methods can be classified into various flowing categories like Non-interference analysis, information flow analysis, Covert Flow Tree method and Shared Resource Matrix method.
[13] "Trends and Challenges in Network Covert Channels Countermeasure"	This paper investigates trends and challenges in the development of countermeasures against popular network covert channels. The findings of this study show that many works are highly specialist, and high-level and general techniques may be advantageous in developing a successful strategy for reducing security concerns brought on by network hidden channels. We learnt that it is easier to block those timing channels that use inter-packet information. However, the indiscriminate penalization of traffic flows is the disadvantage of the strategy. The paper tells us that reducing steganographic bandwidth can help make the

	channel unsuitable for attack. This paper also discusses about the usage of entropy-based schemes which prevents the attacker from limiting the rate to evade detection.

Based on the literature survey, we are focusing on detecting timing channels by adopting the following 8 techniques:

- **Packet presence Technique:** For this method to work, the sender and receiver must be in sync and agree on a set time period for sampling. During a certain interval, a packet's presence or absence translates to a binary 0 or 1 correspondingly.
- **Fixed intervals:** In order to represent 0s and 1s, this approach establishes defined inter-departure times for packets.
- **The Jitterbug** modifies a pre-existing network transmission. In order to create packet inter-departure times divisible by X or X/2, depending on the covert symbol to broadcast, it sets a base sampling time interval of let's say X and adds a little delay to them. Divisibility by X or X/2 may be used to represent binary bits 0 or 1 respectively or vice-versa.
- **Huffman encoding**: This approach is based on the Huffman compression algorithm to directly encode each covert symbol into a group of packets with various packet inter-departure times dependent on the frequency of the encoded symbol.
- **One threshold**: This approach defines a threshold X for packet inter-arrival periods, especially for Android systems with video services. Delays in packet arrivals are recorded as 1s or 0s, depending on if the delay value is less than or greater than X.
- **Packet bursts**: This approach generates packet bursts that are separated apart by a waiting period say, X. The hidden symbol or piece of code to transmit is directly defined by the number of packets in a burst.
- **Differential/derivative**: Every time a '1' is to be transmitted, the basic packet inter-departure time value (idtv) is updated by adding or subtracting t_{diff} from the preceding idtv. If '0' is to be transmitted, the last idtv remains unchanged.

4. PROJECT REQUIREMENTS SPECIFICATION

4.1 INTRODUCTION

This chapter specifies the requirements for "Covert Channel Detection and Prevention" software. The purpose of this software is to detect and prevent covert channels, which are methods used to communicate secretly over computer networks. The intended audience for this SRS includes software developers, quality assurance testers, project managers, and other stakeholders involved in the development and deployment of the software.

4.2 SCOPE

The "Covert Channel Detection and Prevention" software will provide a means for detecting and preventing covert channels in computer networks. While a wide variety of covert channels exist, the focus of this project will be on detecting timing channels. The software will be designed to work with various types of network protocols and will be compatible with a wide range of hardware configurations. In addition to detecting covert timing channels, the software will be tested in a virtual environment using multiple virtual machines (VMs) and data transmission through covert timing channels. The software will also be designed to disrupt covert timing channels after they have been detected.

4.3 FUNCTIONAL REQUIREMENTS

The following are the functional requirements for the "Covert Channel Detection and Prevention" software:

4.3.1 DETECTION OF COVERT TIMING CHANNELS

The software must be capable to detect the presence of covert timing channels in computer networks.

4.3.2 TWO LEVELS OF DETECTION

The software must have two levels of detection. The first level involves a superficial analysis of delay and packet length. The second level involves a more advanced analysis using machine learning to reduce the number of false positives.

4.3.3 COMPATIBILITY WITH BPF

The software must be compatible with BPF (Berkeley Packet Filter) for network packet filtering and analysis.

4.3.4 STORAGE OF PACKETS

The software must store each packet for a given source and destination and only delete them when the suspicion of a covert timing channel has been cleared.

4.3.5 TESTING IN A VIRTUAL ENVIRONMENT

The software must be tested in a virtual environment with multiple virtual machines and data transmission through covert timing channels.

4.3.6 DISRUPTION OF COVERT TIMING CHANNELS

The software must be designed to disrupt covert timing channels after they have been detected. Techniques such as timestamp manipulation and the introduction of random transmission delays of packets should be used.

4.4 NON-FUNCTIONAL REQUIREMENTS

The following are the non-functional requirements for the "Covert Channel Detection and Prevention" software:

4.4.1 RELIABILITY

The software shall have a 99.99% uptime rate in detecting and preventing covert timing channels over a period of one month.

4.4.2 SCALABILITY

The software shall be able to handle a minimum of 25,000 network packets per second without experiencing any degradation in performance.

4.4.3 EASE OF INSTALLATION AND CONFIGURATION

The software installation process shall take no more than 30 minutes, and the configuration process shall take no more than 1 hour, as confirmed by user feedback.

4.4.4 MINIMAL NETWORK PERFORMANCE IMPACT

The software shall have a maximum overhead of 10% on network performance.

4.4.5 ACCURACY IN DETECTING COVERT TIMING CHANNELS

The software shall have a detection rate greater than 90% in detecting covert timing channels when tested in a test environment.

4.5 CONSTRAINTS

The following are the constraints for the "Covert Channel Detection and Prevention" software:

4.5.1 LEGAL COMPLIANCE CONSTRAINT

The software development and operation must comply with all applicable laws and regulations.

4.5.2 BUDGET AND TIMELINE CONSTRAINT

The software must be developed within the specified budget and timeline as agreed upon by the project stakeholders.

4.6 ASSUMPTIONS

The following are the assumptions made during the development of the "Covert Channel Detection and Prevention" software:

4.6.1 LEGITIMATE USE

It is assumed that the software will be used for legitimate purposes only and will not be used to engage in any illegal activities.

4.6.2 AUTHORITY TO MONITOR

It is assumed that the network traffic to be monitored by the software will be within the authority of the network administrator to monitor, and that the network administrator has obtained any necessary legal permissions or consents to monitor such traffic.

4.6.3 HARDWARE AND SOFTWARE REQUIREMENTS

It is assumed that the hardware and software requirements of the target systems, as specified by the software, will be met, including but not limited to processing power, memory, and operating system compatibility.

4.6.4 TEST ENVIRONMENT

It is assumed that the test environment used to evaluate the software's ability to detect covert timing channel transmission will accurately simulate such transmission and will not introduce any external factors that may affect the software's performance.

4.7 DEPENDENCIES

The following are the dependencies for the "Covert Channel Detection and Prevention" software:

4.7.1 BPF DEPENDENCY

The software will be dependent on BPF (Berkeley Packet Filter) for network packet filtering and analysis.

4.7.2 ML LIBRARY DEPENDENCY

The software will be dependent on machine learning (ML) libraries for training and embedding the ML model into the software.

4.7.3 VIRTUAL ENVIRONMENT DEPENDENCY

The software will be dependent on a virtual environment for testing with multiple virtual machines (VMs) to simulate a realistic network environment.

4.7.4 NETWORK PROTOCOL DEPENDENCY

The software will be dependent on network protocols that are supported by the target systems to monitor and analyse network traffic.

4.8 ACCEPTANCE CRITERIA

4.8.1 DETECTION OF COVERT TIMING CHANNELS

The software is capable to detect the presence of covert timing channels in computer networks.

4.8.2 TWO LEVELS OF DETECTION

The software has two levels of detection. The first level involves a superficial analysis of delay and packet length. The second level involves a more advanced analysis using machine learning to reduce the number of false positives.

4.8.3 COMPATIBILITY WITH BPF

The software is compatible with BPF (Berkeley Packet Filter) for network packet filtering and analysis.

4.8.4 STORAGE OF PACKETS

The software stores each packet for a given source and destination and only deletes them when the suspicion of a covert timing channel has been cleared.

4.8.5 TESTING IN A VIRTUAL ENVIRONMENT

The software is tested in a virtual environment with multiple virtual machines and data transmission through covert timing channels.

4.8.6 DISRUPTION OF COVERT TIMING CHANNELS

The software disrupts covert timing channels after they have been detected.

4.8.7 RELIABILITY

The software has a 99.99% uptime rate in detecting and preventing covert timing channels over a period of one month.

4.8.8 SCALABILITY

The software can handle a minimum of 25,000 network packets per second without experiencing any degradation in performance.

4.8.9 EASE OF INSTALLATION AND CONFIGURATION

The software installation process takes no more than 30 minutes, and the configuration process takes no more than 1 hour, as confirmed by user feedback.

4.8.10 MINIMAL NETWORK PERFORMANCE IMPACT

The software has a maximum overhead of 10% on network performance.

4.8.11 ACCURACY IN DETECTING COVERT TIMING CHANNELS

The software has a detection rate greater than 90% in detecting covert timing channels when tested in a test environment.

4.9 GLOSSARY

The following terms are used in this SRS:

1. **BPF** - Berkeley Packet Filter, a system for filtering and analysing network packets.

2. **Covert channel** - a method used to communicate secretly over computer networks.

3. **ML** - Machine Learning, a field of study that uses algorithms to learn patterns and make predictions based on data.

4. **Test environment** - a controlled environment used for testing software or hardware.

5. **Timing channel** - a covert channel that uses variations in timing to transmit information.

4.10 CONCLUSION

This chapter provides a detailed specification of the requirements for the "Covert Channel Detection and Prevention" software. The chapter outlines the functional and non-functional requirements, constraints, assumptions, and dependencies for the software. This chapter will serve as a reference for software developers, quality assurance testers, project managers, and other stakeholders involved in the development and deployment of the software.

5. HIGH-LEVEL DESIGN

5.1 INTRODUCTION

This chapter describes the general architecture and parts of the software tool we plan to develop. It gives a broad overview of the functioning of the solution, outlining its goals, demands, and design tenets.

An overview of the software solution, its functional requirements, and its general architecture, including the parts, modules, and interfaces needed to offer the desired functionality, are typically included in this document. For developers, stakeholders, and project managers, it acts as a roadmap for comprehending the scope and direction of the software solution.
The solution's high-level design will be based on-

• **Network Packet Capture and Filtering:** This component utilizes BPF to capture and filter network packets depending on their source and destination addresses.

• **Superficial Detection**: To detect any irregularities that would point to the presence of a covert timing channel, this component does a superficial examination of the filtered packets by examining the delay and packet length.

• **ML-based Pattern Analysis**: This component uses a trained ML model to conduct a more thorough analysis of the packets that make it beyond the superficial detection stage in order to find patterns that might point to the existence of a covert timing channel.

• **Packet Storage**: Until the possibility of a covert timing channel is disproved, this component records each packet for a specific source and destination

•**Virtual Environment**: To confirm the software solution's effectiveness in real-world circumstances, this component evaluates it in a virtual environment with several VMs and data transmission using covert timing channels.

• **Disruptive Measures**: This element is intended to stop the transmission of covert timing channels after they have been identified.

• **Test Environment**: To assess the accuracy of the software, this component simulates covert timing channel transfer between client and server.

5.2 CURRENT IMPLEMENTATION

The current systems that are used in detecting and preventing timing covert channels are:

- **Network Intrusion Detection Systems (NIDS)**: NIDS are programs that keep an eye on network activity for any indications of hostile activity, including hidden channels. To identify abnormal network activity and notify security professionals, they employ a variety of detection techniques, such as pattern matching and anomaly detection.

- **Firewall rules and access controls:** They are used to restrict the kinds of network traffic that are permitted to transit across a network. It is more difficult to exfiltrate data from the network by blocking traffic that is known to be linked with covert channels.

5.3 DESIGN CONSIDERATIONS

5.3.1 DESIGN GOALS

- **Accuracy**: With a minimum of false positives, the software solution should be able to detect concealed timing channels in network traffic.
- **Effectiveness**: With minimal negative effects on network performance, the software solution should be able to identify covert timing channels in real time.
- **Compatibility**: In order to filter and analyze network packets, the software solution must be compatible with BPF.
- **Flexibility**: The software solution must be flexible enough to accommodate various network setups and offer a range of detectable threats.
- **Data Storage**: Once the possibility of a covert timing channel has been eliminated, the software solution should store each packet for a specific source and destination.

- **Disruptive Capabilities**: The software solution should be designed to disrupt covert timing channels after they have been detected.
- **Testability**: The software solution should use a test environment for simulating covert timing channel transmission between client and server

5.3.2 ARCHITECTURE CHOICES

We have considered a number of architectural options, such as –

Detection Methods: We have considered several methods, such as statistical analysis, deep packet inspection, and behavioural analysis, for discovering covert timing channels. We decided to combine superficial detection by analyzing delay and packet length with ML-based pattern analysis to minimize false positives after weighing the advantages and disadvantages of each method. This method balances accuracy and efficiency while having the least possible negative effects on network performance.

Benefits: This method effectively locates covert timing channels, offers a high level of accuracy, and reduces false positives. Additionally, it is effective and barely affects network performance.

Disadvantages: ML-based pattern analysis requires a large amount of training data to build an effective model, which can be time-consuming and resource-intensive.

Disruptive Capabilities: We considered different options for how our software solution could disrupt covert timing channels after they were detected, including blocking suspicious traffic, modifying packet timing or sequence, and delaying packets. Ultimately, we chose to delay packets by introducing random jitter into the packet transmission to disrupt the covert timing channel.

Pros: This approach is effective in disrupting covert timing channels and has a low impact on network performance. It also allows for the detection of the covert timing channel to continue, providing additional data for analysis.

Cons: Delaying packets can potentially affect network performance, particularly for real-time applications

5.3.3 CONSTRAINTS ASSUMPTIONS AND DEPENDENCIES

- **Interoperability Requirements**: To filter and analyze network packets, our software solution must be compatible with BPF. This presupposes that the system will be installed in a setting that allows for the use of BPF.

- **Interface/Protocol Requirements**: To analyze network traffic, our software solution needs network traffic data. The use of the BPF filtering and analysis tools is predicated on the network traffic being in a standard format and being able to be recorded and examined

- **Performance-Related Problems**: The volume of network traffic being analyzed and the difficulty of the machine learning models being utilized for analysis may have an impact on the performance of our software solution. When analyzing large amounts of traffic, the system might also encounter delays or bottlenecks.

- **End-User Environment**: Our software solution assumes that end users have a fundamental understanding of machine learning and network traffic analysis ideas. It also presumes that end users will have access to the hardware and software resources required for system deployment and operation.

- **Availability of Resources**: To analyze network traffic and build machine learning models, our software solution may need a lot of computer power. This presupposes the availability of the required resources, including computing power, memory, and storage.

- **Hardware or Software Environment:** Our software solution can require operating systems or programming languages, as well as other hardware or software. When deploying the system, these dependencies must be considered.

5.4 HIGH LEVEL SYSTEM DESIGN

The high-level design document of the software identifies the logical user groups, application components, data components, and interfacing systems.

5.4.1 LOGICAL GROUPS -

Network administrators

- Security analysts
- System operators

5.4.2 APPLICATION COMPONENTS:

- Packet capture module
- Covert timing channel detection module
- Covert timing channel disruption module
- Machine learning-based detection module.
- Database module for storing packets.

5.4.3 DATA COMPONENTS:

- Captured network packets.
- Training dataset for machine learning module.

5.4.4 INTERFACING SYSTEMS:

- BPF (Berkeley Packet Filter) for network packet filtering and analysis
- Virtual environments for testing and simulation

We also consider how the components interact with each other to analyse the dependencies between various components.

5.4.5 COLLABORATION BETWEEN COMPONENTS:

- The packet capture module captures network packets and sends them to the covert timing channel detection module.

- The covert timing channel detection module analyzes the captured packets using superficial detection techniques like delay and packet length analysis, and machine learning-based pattern analysis to detect the presence of covert timing channels.
- If a covert timing channel is detected, the covert timing channel disruption module disrupts the channel by manipulating the network traffic.
- The database module stores all the packets captured by the system until they are cleared of suspicion.
- The machine learning-based detection module uses a training dataset to continuously improve the accuracy of detection.
- The system interfaces with BPF for packet analysis and virtual environments for testing and simulation.

5.4.6 PATTERNS USED

- **Observer pattern**: This pattern can be used to allow different components of the system to observe the network traffic and detect any covert timing channels. The packet capture module can be the subject, and the detection modules can be the observers.

- **Factory pattern:** This pattern can be used to create instances of the different detection modules based on the level of detection required. The factory can take in input parameters to determine which module to instantiate.

5.5 MASTER CLASS DIAGRAM

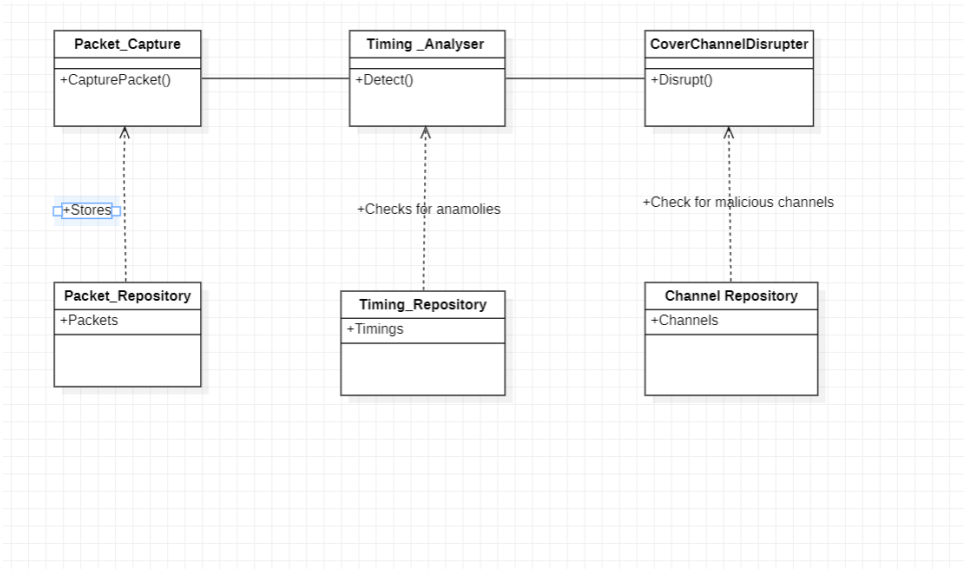

Figure 1: Master Class Diagram

5.6 REUSABILITY CONSIDERATIONS

Reusability considerations are an important aspect of any software development project. In our project, we have identified the following components that can be reused:

- **Algorithms for detecting timing-based covert channels**: We intend to create algorithms for detecting timing-based covert channels in a variety of applications. Other security applications where timing analysis is crucial can make use of these algorithms.

- **Test suites**: To evaluate the detection methods, we will create a number of test suites. Other timing analysis tools can be tested using these test suites.

- **Logging and reporting modules**: To record and report the discovered covert channels, we will create logging and reporting modules. Other security applications that need logging and reporting features can reuse these modules.

- **User interface components**: We will develop a user interface that allows users to configure the detection algorithms and view the detected covert channels. These user interface components can be reused in other applications that require a similar user interface.

In addition to these components, we will also consider using available reusable components, such as libraries and frameworks, to minimize development effort and improve the quality of the system. We will evaluate these reusable components based on their compatibility with our project requirements, ease of integration, and licensing restrictions.

5.7 STATE DIAGRAM

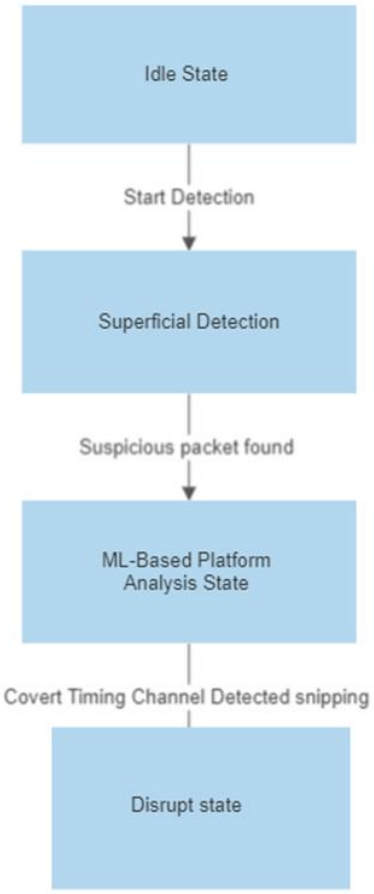

Figure 2: State Diagram

5.8 EXTERNAL INTERFACES DIAGRAM

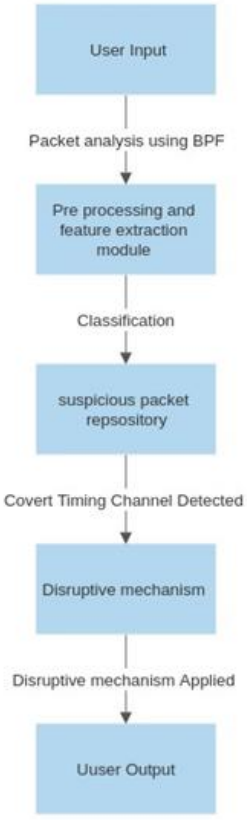

Figure 3: External Interfaces Diagram

5.9 USER INTERFACE

User Interface The user interface would consist of –

• A dashboard showing the overall status of the network, including any detected covert timing channels.

• A settings page where the user can configure the software to their needs, such as the level of detection they prefer, the type of analysis to be performed, and the packet filtering criteria.

• A page for displaying the details of individual packets, including the source and destination addresses, time stamps, and any relevant packet data.

• A page for displaying alerts and notifications generated by the software, indicating the presence of a potential covert timing channel.

• A page for configuring and training the ML model used for pattern analysis.

• A page for simulating covert timing channel transmissions and testing the software's detection and disruption capabilities.

• A user-friendly interface that allows users to easily navigate and interact with the software, including the ability to drill down into individual packets, view detailed reports, and access relevant documentation and support resources.

5.10 PACKAGING AND DEPLOYMENT DIAGRAM

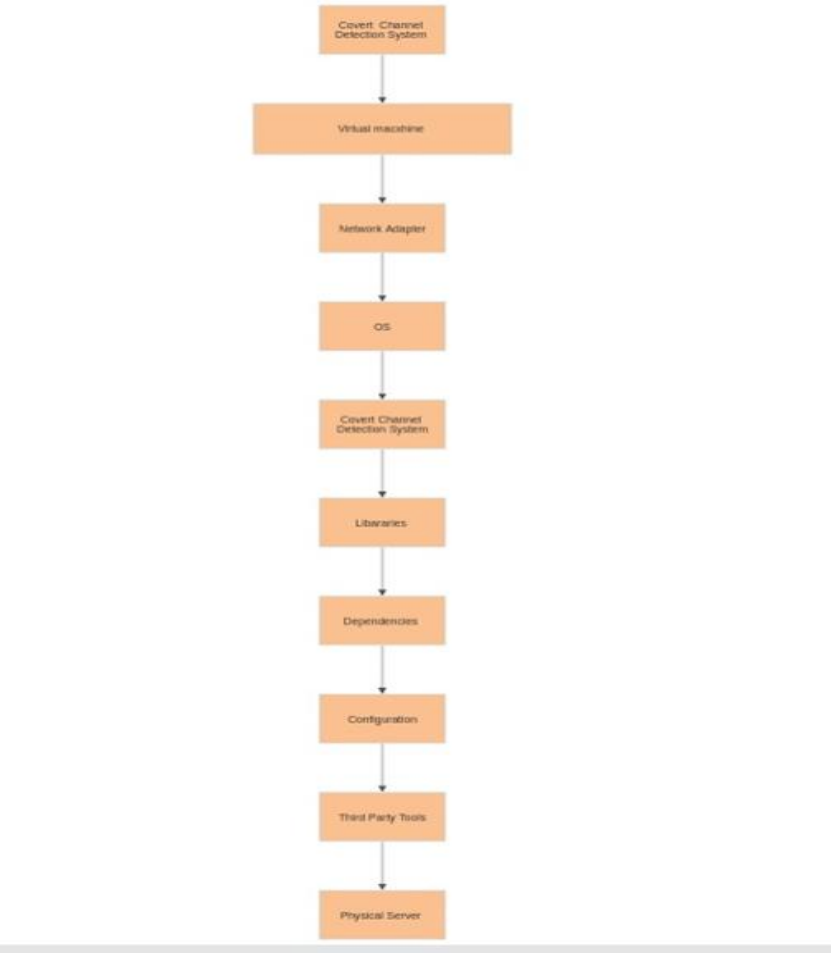

Figure 4: Packaging and Deployment Diagram

5.11 DESIGN DETAILS

Based on the project requirements and design goals, the system may depend on various platforms, systems, and processes. Some of the important ones are:

- **Operating System:** The system may depend on a specific operating system such as Windows, Linux, or MacOS.
- **Programming Language**: The system may require a specific programming language such as Java, Python, or C++.
- **Database Management System:** The system may depend on a specific database management system such as MySQL, Oracle, or MongoDB.
- **Web Server**: The system may require a web server such as Apache or Nginx.
- **Network Infrastructure:** The system may depend on a reliable network infrastructure to ensure proper communication between the system components.
- **Security Infrastructure**: The system may require a security infrastructure such as firewalls, encryption, or multi-factor authentication to ensure the security of the system.

6. DETAILED DESIGN

- **6.1 SYSTEM ARCHITECTURE**: The software will be designed using a client-server architecture. The client will be responsible for capturing network packets and sending them to the server for analysis. The server will be responsible for analyzing the packets and detecting covert timing channels.

- **6.2 COVERT TIMING CHANNEL DETECTION:** The software will provide two levels of detection to identify covert timing channels. The first level of detection will involve superficial detection by analyzing delay and packet length. The second level will involve ML-based pattern analysis to reduce false positives. For ML-based pattern analysis, a dataset of training models will be created, and the ML model will be embedded in the software to analyze packets.

- **6.3 BPF CODE FOR PACKET ANALYSIS**: The software will use BPF code for network packet filtering and analysis. BPF code will be integrated into the software to allow for network packet filtering and analysis.

- **6.4 PACKET STORAGE:** The software will store each packet for a given source and destination and only delete the packets once the suspicion of a covert timing channel has been cleared. The packet storage will be done on the server-side.
- **6.5 TESTING ENVIRONMENT**: The software will be tested in a virtual environment with multiple VMs and data transmission through covert timing channels. A test environment will be set up to simulate covert timing channel transmission between the client and server.
- **6.6 COVERT TIMING CHANNEL DISRUPTION:** The software will be designed to disrupt covert timing channels after they have been detected. Once a covert timing channel is detected, the software will take the necessary actions to disrupt the communication.

- **6.7 COMPATIBILITY**: The software will be compatible with different operating systems and network configurations.
- **6.8 SECURITY**: The software will ensure the security of data being transmitted, stored, and analysed. Appropriate encryption mechanisms will be incorporated to ensure secure transmission and storage of data.
- **6.9 USER INTERFACE** : The software will have a user-friendly interface that will allow users to initiate packet capture, view captured packets, and detect covert timing channels.

7. IMPLEMENTATION

As mentioned in the literature survey, the tool we aim to develop will try to detect 8 different types of timing covert channels available across the literature. Implementation of those 8 types of channels is necessary for the proper development of the tool. As of now, we have successfully implemented 3 of the planned covert channel communication techniques.

7.1 ONE THRESHOLD TECHNIQUE

7.1.1 CLIENT-SIDE ALGORITHM

1. Take the data to be sent as input from a file/user input in ASCII format.
2. Convert the data to binary bits.
3. Construct a UDP packet with the destination IP and Port of the Server
4. If the bit to be sent is “0”, send the packet immediately.
5. If the bit to be sent is “1”, send the packet after a predefined interval.

7.1.2 SERVER-SIDE ALGORITHM

1. Create an empty bitstring.
2. Accept the packets coming from the client side and associate an arrival timestamp with each packet.
3. If the time difference between 2 consecutive packets is less than the predefined interval, append “0” to the bitstring, else append “1”.
4. Convert bitstring back to ASCII format.

7.2 TIMESTAMP MANIPULATION

7.2.1 CLIENT-SIDE ALGORITHM

1. Take the data to be sent as input from a file/user input in ASCII format.
2. Convert the data to binary bits.
3. Create a TCP connection to the server side.
4. If the bit to be sent is “0”, wait till the LSB TCP timestamp is even and then send the packet.

5.If the bit to be sent is “1”, wait till the LSB TCP timestamp is odd and then send the packet.

7.2.2 SERVER-SIDE ALGORITHM

1.Create an empty bitstring.

2.Accept the packets coming from the client side.

3.If the LSB of the TCP timestamp is even, append “0” to the bitstring, else append “1”.

4.Convert bitstring back to ASCII format.

7.3 PACKET BURST ENCODING

7.3.1 CLIENT-SIDE ALGORITHM

1.Take the data to be sent as input from a file/user input in ASCII format.

2.Convert the data to a binary bitstring.

3.Split the bitstring into small bitstrings of length 4.

4.Create a TCP connection to the server side.

6.Convert each small binary bitstring to integers between 0 and 15 say X.

7.For every interval of 5 seconds, send X+1 packets, corresponding to each binary bitstring.

7.1.2 SERVER-SIDE ALGORITHM

1.Create an empty bitstring.

2.Accept the packets coming from the client side.

3.Store the incoming packets in the pandas' data frame along with their arrival time.

4.Once the TCP connection is terminated, use hierarchical clustering using the ward’s linkage to cluster the packets based on arrival time.

5.Calculate the number of packets in each cluster say Y.

6.Convert integer Y-1 to into binary and append it to the bitstring for every cluster.

7.Convert bitstring back to ASCII format.

8. CONCLUSION: CAPSTONE PROJECT PHASE–1

At the end of phase 1 of the capstone project, we have achieved various objectives.

1. We were able to formulate a well-defined problem statement that is developing a tool for detecting covert timing channels in the network.
2. We surveyed various creation, detection and prevention techniques for covert channels available across the literature.
3. We conducted an in-depth feasibility study for the project.
4. We were able to identify publicly available datasets for covert channel detection.
5. We defined various functional and non-functional requirements for the tool which we aim to develop.
6. We were able to put forward a high-level and a detailed design document.
7. We implemented some of the covert communication techniques that the tool will try to detect and disrupt.

9. PLAN OF WORK: CAPSTONE PROJECT PHASE–2

1. Indigenous Dataset generation for covert channel detection.
2. Training ML model for detection of the covert channel using the generated dataset.
3. Designing the test environment for evaluating the performance of the covert channel detection tool.
4. Development of the covert channel detection tool.
5. Designing the test environment for evaluating the performance of the covert channel prevention tool.
6. Development of the covert channel prevention tool.
7. Integration of the covert channel detection and prevention tools with the test environment.
8. Testing the integrated system for detecting and preventing covert channels in different scenarios.
9. Fine-tuning the ML model and updating the detection tool based on the performance in the test environment.
10. Documenting the final system architecture, design, implementation, and testing results.
11. Release of the covert channel detection and prevention tool as a software product and research artifact.

10. REFERENCES

[1] Marco Zuppelli, Luca Caviglione, Wojciech Mazurczyk, Andreas Schaffhauser, Matteo Repetto, "Code Augmentation for Detecting Covert Channels Targeting the IPv6 FLow Label," 2021, pp. 450-456

[2] Matteo Repetto, Luca Caviglione, Marco Zuppelli, "bccstego: A Framework for Investigating Network Covert Channels," 2021.

[3] Félix Iglesias, Fares Meghdouri, Robert Annessi, Tanja Zseby, "CCgen: Injecting Covert Channels into Network Traffic," 2022, pp. 1-11.

[4] Marco Zuppelli, Luca Caviglione, "pcapStego: A Tool for Generating Traffic Traces for Experimenting with Network Covert Channels," 2021.

[5] Félix Iglesias, Robert Annessi, T. Zseby, "DAT detectors: uncovering TCP/IP covert channels by descriptive analytics," 2016, pp. 3011-3029.

[6] Muawia A. Elsadig, Yahia A. Fadlalla, "Packet Length Covert Channel: A Detection Scheme," 2021 pp. 1-7.

[7] Arti Dua, Vinita Jindal, Punam Bedi, "Covert Communication using Address Resolution Protocol Broadcast Request Messages," 2021, pp. 1-6.

[8] Muawia A. Elsadig, Ahmed Gafar, "Covert Channel Detection: Machine Learning Approaches," vol 10, pp. 38391-38405, 2022.

[9] Marco Zuppelli, Matteo Repetto, Andreas Schaffhause, Wojciech Mazurczyk, "Code Layering for the Detection of Network Covert Channels in Agentless Systems", vol. 19, no. 3, pp. 2282- 2294, 2022.

[10] Punam Bedi, Arti Dua, "ARPNetSteg: Network Steganography Using Address Resolution Protocol," vol. 66, no. 4, pp. 671-677, 2020.

[11] Félix Iglesias, Valentin Bernhardt, Robert Annessi, Tanjab Zseby, "Decision Tree Rule Induction for Detecting Covert Timing Channels in TCP/IP Traffic," {*1st International Cross-Domain Conference for Machine Learning and Knowledge Extraction(CD-MAKE)*}, pp. 105-122, Aug 2017.

[12] Muawia A. Elsadig, Yahia A. Fadlalla, "Network Protocol Covert Channels: Countermeasures Techniques," 2017.

[13] Caviglione, L. “Trends and Challenges in Network Covert Channels Countermeasures”. Appl. Sci. 2021, 11, 1641.

[14] Dewank Pant, Manon Wason, Jibraan Singh Chahal, "Cross VM Covert Channel Implementation", 2018.

APPENDIX A : DEFINITIONS , ACRONYMS AND ABBREVIATIONS

- ıHLD - High Level Design
- UML - Unified Modeling Language
- API - Application Programming Interface
- UI - User Interface
- UX - User Experience
- DB - Database
- CRUD - Create, Read, Update, Delete
- . MVP - Minimum Viable Product
- QA - Quality Assurance 1
- API - Application Programming Interface
- AWS - Amazon Web Services
- SaaS - Software as a Service
- HTTP - Hypertext Transfer Protocol
- SQL - Structured Query Language

APPENDIX B :REFERNCES

- G. Vigna and W. Robertson, "Detecting covert timing channels," 2011 IEEE International Conference on Communications (ICC), Kyoto, 2011, pp. 1-6, doi: 10.1109/icc.2011.5962938.

- L. Adamic, E. Adar, and P. Resnick, "The small world web," in Proceedings of the First ACM Conference on Electronic Commerce, Denver, Colorado, USA, 1999, pp. 44-54, doi: 10.1145/336992.337012.

- "BPF and libpcap," The Tcpdump Group, 2021. [Online]. Available: https://www.tcpdump.org/.

- "Scikit-learn: Machine Learning in Python," scikit-learn developers, 2021. [Online]. Available: https://scikit-learn.org/stable/.

- "PyQt: Python GUI Programming - Learn PyQt Toolkit," Qt Company, 2021. [Online]. Available: https://www.qt.io/qt-for-python.

APPENDIX C: RECORD OF CHANGE OF HISTORY

#	Date	Document Version No.	Change Description	Reason for Change
1.	**1-5-23**	**1**	-	**-**
2.				
3.				