***Logo, company name

Description automatically generated***

***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**  **Kumkum Geervani**  **Ghanshyam Bhat**  **Prajwal Bhat** | **PES1UG20CS193**  **PES1UG20CS184**  **PES1UG20CS153**  **PES1UG20CS290** |

*Under the guidance of*

|  |
| --- |
| **Dr. Sapna V M**  Associate 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

Logo, company name

Description automatically generated

**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*

**‘Covert Channel Detection and Prevention’**

*is a bonafide work carried out by*

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

in partial fulfilment for the completion of sixth semester Capstone Project Phase - 1 (UE20CS390A) 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  **<Name of the Guide>**  Designation | Signature  Dr. Shylaja S S  Chairperson | Signature  Dr. B K Keshavan  Dean of Faculty |

**External Viva**

|  |  |
| --- | --- |
| **Name of the Examiners**  **1.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Signature with Date**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_    **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

**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 2023. 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 UE20CS390A - 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. Therefore, our goal is to create a software tool that can identify and stop the use of covert channels.

This project aims to detect and prevent network timing covert channels in computer networks with the help of software. This project will include various tools, machine learning algorithms, and methodologies all aiming to detect and prevent such channels. A well-developed software is designed to support a variety of network protocols and a variety of network setups. Various machine learning techniques are combined to build a model to train our data and classify the channel as a covert channel or not.

**TABLE OF CONTENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chapter No.** | | **Title** | **Page No.** | |
|  | **INTRODUCTION** | | | 1 |
|  | **PROBLEM STATEMENT** | | | 2 |
|  | **LITERATURE SURVEY** | | | 3 |
| **4.** | **PROJECT REQUIREMENT SPECIFICATIONS** | | | 10-16 |
|  | **4.1 INTRODUCTION** | | | 10 |
|  | **4.2 SCOPE** | | | 10 |
|  | **4.3 FUNCTIONAL REQUIREMENTS**   |  |  | | --- | --- | | **4.3.1** | **DETECTION OF COVERT TIMING CHANNELS** | | **4.3.2** | **TWO LEVELS OF DETECTION** | | **4.3.3** | **COMPATIBILITY WITH BPF** | | **4.3.4** | **STORAGE OF PACKETS** | | **4.3.5** | **TESTING IN A VIRTUAL ENVIRONMENT** | | **4.3.6** | **DISRUPTION OF COVERT TIMING CHANNELS** | | | | 10-11 |
|  | **4.4 NON-FUNCTIONAL REQUIREMENTS**   |  |  | | --- | --- | | **4.4.1** | **RELIABILITY** | | **4.4.2** | **SCALABILITY** | | **4.4.3** | **EASE OF INSTALLATION AND CONFIGURATION** | | **4.4.4** | **MINIMAL NETWORK PERFORMANCE IMPACT** | | **4.4.5** | **ACCURACY IN DETECTING COVERT TIMING CHANNELS** | | | | 11-12 |
|  | **4.5 CONSTRAINTS**   |  |  | | --- | --- | | **4.5.1** | **LEGAL COMPLIANCE CONSTRAINT** | | **4.5.2** | **BUDGET AND TIMELINE CONSTRAINT** | | | | 12 |
|  | **4.6 ASSUMPTIONS**   |  |  | | --- | --- | | **4.6.1** | **LEGITIMATE USE** | | **4.6.2** | **AUTHORITY TO MONITOR** | | **4.6.3** | **HARDWARE AND SOFTWARE REQUIREMENTS** | | **4.6.4** | **TEST ENVIRONMENT** | | | | 12-13 |
|  | **4.7 DEPENDENCIES**   |  |  | | --- | --- | | **4.7.1** | **BPF DEPENDENCY** | | **4.7.2** | **ML LIBRARY DEPENDENCY** | | **4.7.3** | **VIRTUAL ENVIRONMENT DEPENDENCY** | | **4.7.4** | **NETWORK PROTOCOL DEPENDENCY** | | | | 14 |
|  | **4.8 ACCEPTANCE CRITERIA** | | | 15 |
|  | **4.9 GLOSSARY** | | | 16 |
|  | **4.10 CONCLUSION** | | | 16 |
| **5.** | **HIGH-LEVEL DESIGN** | | | 17-29 |
|  | **5.1 INTRODUCTION** | | | 17 |
|  | **5.2 CURRENT IMPLEMENTATION** | | | 18 |
|  | **5.3 DESIGN CONSIDERATIONS**   |  |  | | --- | --- | | **5.3.1** | **DESIGN GOALS** | | **5.3.2** | **ARCHITECTURE CHOICES** | | **5.3.3** | **CONSTRAINTS ASSUMPTIONS AND DEPENDENCIES** | | | | 19-21 |
|  | **5.4 HIGH-LEVEL SYSTEM DESIGN**   |  |  | | --- | --- | | **5.4.1** | **LOGICAL GROUPS** | | **5.4.2** | **APPLICATION COMPONENTS** | | **5.4.3** | **DATA COMPONENTS** | | **5.4.4** | **INTERFACING SYSTEMS** | | **5.4.5** | **COLLABORATION BETWEEN COMPONENTS** | | **5.4.6** | **PATTERNS USED** | | | | 21-23 |
|  | **5.5 MASTER CLASS DIAGRAM** | | | 24 |
|  | **5.6 REUSABILITY CONSIDERATIONS** | | | 24-25 |
|  | **5.7 STATE DIAGRAM** | | | 26 |
|  | **5.8 EXTERNAL INTERFACES DIAGRAM** | | | 27 |
|  | **5.9 USER INTERFACE** | | | 27-28 |
|  | **5.10 PACKAGING AND DEPLOYMENT DIAGRAM** | | | 28 |
|  | **5.11 DESIGN DETAILS** | | | 29 |
| **6.** | **DETAILED DESIGN** | | | 28-31 |
|  | **6.1 SYSTEM ARCHITECTURE** | | | 30 |
|  | **6.2 COVERT TIMING CHANNEL DETECTION** | | | 30 |
|  | **6.3 BPF CODE FOR PACKET ANALYSIS** | | | 30 |
|  | **6.4 PACKET STORAGE** | | | 30 |
|  | **6.5 TESTING ENVIRONMENT** | | | 30 |
|  | **6.6 COVERT TIMING CHANNEL DISRUPTION** | | | 31 |
|  | **6.7 COMPATIBILITY** | | | 31 |
|  | **6.8 SECURITY** | | | 31 |
|  | **6.9 USER INTERFACE** | | | 31 |
| **7.** | **IMPLEMENTATION** | | | 32-33 |
|  | **7.1 ONE THRESHOLD TECHNIQUE**   |  |  | | --- | --- | | **7.1.1** | **CLIENT-SIDE ALGORITHM** | | **7.1.2** | **SERVER-SIDE ALGORITHM** | | | | 32 |
|  | **7.2 TIMESTAMP MANIPULATION**   |  |  | | --- | --- | | **7.2.1** | **CLIENT-SIDE ALGORITHM** | | **7.2.2** | **SERVER-SIDE ALGORITHM** | | | | 32-33 |
|  | **7.3 PACKET BURST ENCODING**   |  |  | | --- | --- | | **7.3.1** | **CLIENT-SIDE ALGORITHM** | | **7.3.2** | **SERVER-SIDE ALGORITHM** | | | | 33 |
| **8.** | **CONCLUSION OF CAPSTONE PROJECT PHASE-1** | | | 34 |
| **9.** | **PLAN OF WORK FOR CAPSTONE PROJECT PHASE-2** | | | 35 |
| **10.** | **REFERENCE/ BIBLIOGRAPHY** | | | 36-37 |
| **APPENDIX A: DEFINITIONS, ACRONYMS AND ABBREVIATIONS** | | | | 38 |
| **APPENDIX B: USER MANUAL (OPTIONAL)** | | | | 38-39 |
| **APPENDIX C: RECORD OF CHANGE OF HISTORY** | | | | 39 |

**LIST OF FIGURES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Figure No.** | | **Title** | **Page No.** | |
| **Figure 1** | **MASTER CLASS DIAGRAM** | | | 23 |
| **Figure 2** | **STATE DIAGRAM** | | | 24 |
| **Figure 3** | **EXTERNAL INTERFACES DIAGRAM** | | | 25 |
| **Figure 4** | **PACKAGING AND DEPLOYMENT DIAGRAM** | | | 26 |

**LIST OF TABLES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table No.** | | **Title** | **Page No.** | |
| **Table 1** | **PAPERS SURVEYED AND THEIR TAKEAWAYS** | | | 3 |

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**

In the era of next-generation firewalls, being efficient in firewall configuration is the need of the hour for an organisation. This includes setting up firewall rules and access policies. Even if the firewalls are correctly configured, there is a high probability that a malware might go undetected inside the network.

Many of these malwares try to exfiltrate the organizations’ data covertly. Malicious software often uses covert timing channels to covertly exfiltrate data as detection of timing covert channels is comparatively harder than other types of covert channel mechanisms.

One such trojan is “Pingback” which uses ICMP packets to communicate with a remote server and covey information with the timing of the packets. Another example of such malware is “Stegoloader”. “Stegoloader” is a trojan which embeds sensitive data into images and transmits it to a remote server using timing covert channels.

Thus, 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” | The method for identifying covert channels that target network headers using Bccstego, an inspection framework, is described in this study for investigating network covert channels. Though the method described in this paper proves to provide high throughput, deep packet inspection poses scalability issues. This method neglects the state of the network connection between endpoints, hence aiding 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 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 initiate several 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” | In this study, PcapStego, a tool for creating network covert channels within .pcap files, is discussed. The tool has two primary applications:  1)Large real-world trace generation  2) produces "replayable" conversations that are useful for doing pen testing operations or simulating attacks.  Additionally, this tool aids in the generation of data for IPv6 channels with various levels of complexity and cover channel embedding. |
| [5] “DAT detectors: uncovering TCP/IP covert channels by descriptive analysis” | This paper introduces us to Descriptive Analytics of Traffic(DAT) based detectors 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 discusses packet-length covert channels, which generate covert traffic that resembles ordinary traffic and make it very challenging to detect such channels. With a high degree of accuracy, this technique can find hidden channels. This study reported a 98% accuracy rate and a 0.02 false positive rate. The approach is incompatible with other methods that employ carriers other than packet length since it uses packet length as a carrier. It is not necessary for two communicating entities to exchange shared keys while using the packet-length covert channel. This is more secure and lowers overhead. Complexity and performance overhead are increased. |
| [7] "Covert Communication using Address Resolution Protocol Broadcast Request Messages” | To interact covertly across a local area network, the technique described in this paper uses broadcast request signals from the Address Resolution Protocol (ARP). The suggested method increases the imperceptibility of the secret message by encrypting the secret data with a common seed value that is already known to both the sender and receiver. Instead of explicitly encoding and enhancing the imperceptibility of secret data in the Target Protocol Address field, this approach uses random integers and ASCII code. Frequency analysis is made challenging using various encoding values for the same occurrences of any character. This implies that the ARP Broadcast Request messages are only created and sent by the covert sender for covert communications. |
| [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 presents the idea of the network covert channel triangle (DSM) and talks about how to identify and stop covert channels. Network protocol also referred to as a structured carrier, is utilised as a carrier in a network hidden channel. The paper discusses a variety of detecting techniques. The non-interference analysis, information flow analysis, covert flow tree method, and shared resource matrix method are only a few examples of the different flowing categories into which these techniques can be categorised. |
| [13] “Trends and Challenges in Network Covert Channels Countermeasure” | This paper examines patterns and difficulties in creating defences against well-known network covert channels. The results of this study demonstrate that many works are highly specialised, and high-level and general strategies may be helpful in creating an effective plan for lowering security issues caused by network hidden channels. We discovered that timing channels that rely on inter-packet information are simpler to block. The strategy's drawback is the indiscriminate way in which traffic flows are penalised. According to the paper, lowering the steganographic bandwidth can assist render the channel ineffective for assault. The use of entropy-based techniques, which prohibit attackers from reducing the rate to circumvent detection, is also covered in this study. |

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:** To represent 0s and 1s, this approach establishes defined inter-departure times for packets.
* **The Jitterbug** modifies a pre-existing network transmission. 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 slight time gap between 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 bursts in packets that are separated apart by a waiting period say, X. The hidden symbol to transmit is directly defined by the number of packets in 1 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 tdiff from the preceding idtv. If '0' is to be transmitted, the last idtv remains unchanged.

**4. PROJECT REQUIREMENTS SPECIFICATION**

**4.1 INTRODUCTION**

The prerequisites for "Covert Channel Detection and Prevention" software are outlined in this chapter. This software's job is to find and stop covert channels, which are ways of communicating covertly via networks of computers. The target market for this would be majorly product developers, quality assurance testers, project managers, and other parties involved in the creation and implementation of the product are all included in SRS.

**4.2 SCOPE**

Increasingly, network timing covert channels are being used to deliver malware stealthy features like data exfiltration or covert control of botnet nodes. Because hidden communication can conceal data leaks and virus communication, two crucial aspects of cybercrime, governments and businesses are very concerned about it. "Covert Channel Detection and Prevention" software is a method for identifying and obstructing the timing of covert channels in computer networks. The program aids in discovering unauthorized communication through network traffic and protocol analysis. It helps in categorizing and locating network patterns like covert channel patterns. The program will be designed to function with a range of network protocols and network configurations. The program will also be designed to shut down covert timing channels once they have been found.

**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. The software must then be able to classify the covert channels.

**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 able to run ML models and Python scripts quickly and without much delay. The software must prove its compatibility with BPF filers.

**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**

* The software should be able to detect the presence of covert timing channels in computer networks accurately.
* The software should be able to run machine learning models and Python scripts without significant delay.
* The software should store each packet for a given source and destination and delete them only when the suspicion of a covert timing channel has been cleared.
* The software should be user-friendly and provide a graphical user interface for ease of use.
* The software should be reliable, robust, and secure, with minimal false positives and negatives.
* The software should meet all applicable industry standards and regulations.

**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**

The specifications for the "Covert Channel Detection and Prevention" software are provided in depth in this chapter. The chapter lists the constraints, assumptions, and dependencies for the software's functional and non-functional requirements. The contents of this chapter can be effectively utilized by 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 part of the report 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 by the system by adding random jitter in packet transmission.

**5.2 CURRENT IMPLEMENTATION**

The following systems are currently used in timing covert channels detection and prevention:

**Network Intrusion Detection Systems**: These systems are installed internally to a network and are currently being used to monitor the network traffic for any malicious activities and identify certain patterns including covert channels. Such systems generally employ detection mechanisms like anomaly and signature detections to identify unusual network activities and alert users.

**Access controls and firewall rules** are used to limit the types of network traffic that are allowed to pass across a network. By preventing traffic that is known to be associated with covert channels, data exfiltration from the network is made more challenging.

**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 cleared, the software solution should discard the stored packets which are not flagged for being malicious.
* **Disruptive Capabilitie**s: 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 several 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 have 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**: We aim to build the tool in a way that effectively locates covert timing channels, offers a high level of accuracy, and reduces false positives. Additionally, we also aim that the tool 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 basic understanding of machine learning and network traffic analysis ideas to deploy the software in the accurate environment. 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
* Government Agencies
* Corporate Companies to keep company data safe from exfiltration

**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

**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.
* **Singleton pattern:** This pattern can be used in order to define a single instance for each of the processes in the system and provide global access to it so that the coordination between processes becomes simpler.

**5.5 MASTER CLASS DIAGRAM**

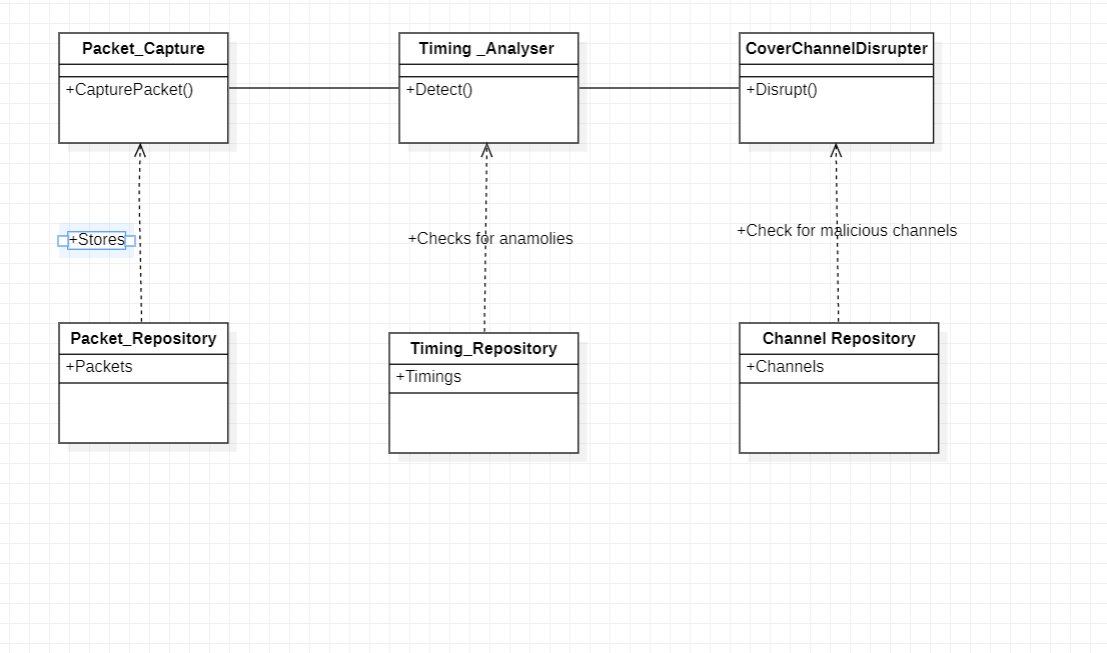


Figure 1: Master Class Diagram

**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**

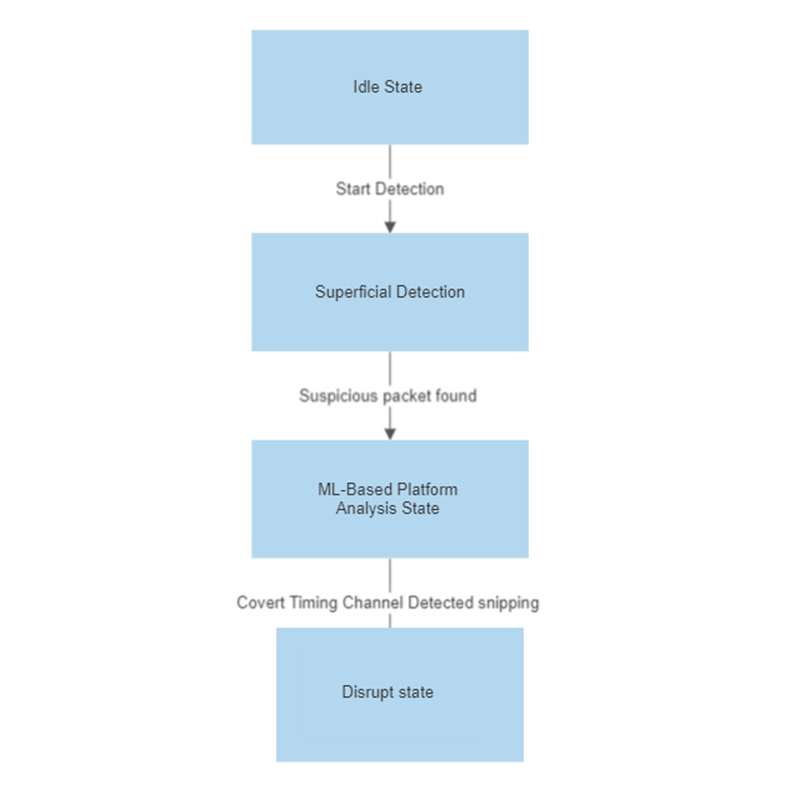


Figure 2: State Diagram

**5.8 EXTERNAL INTERFACES DIAGRAM**

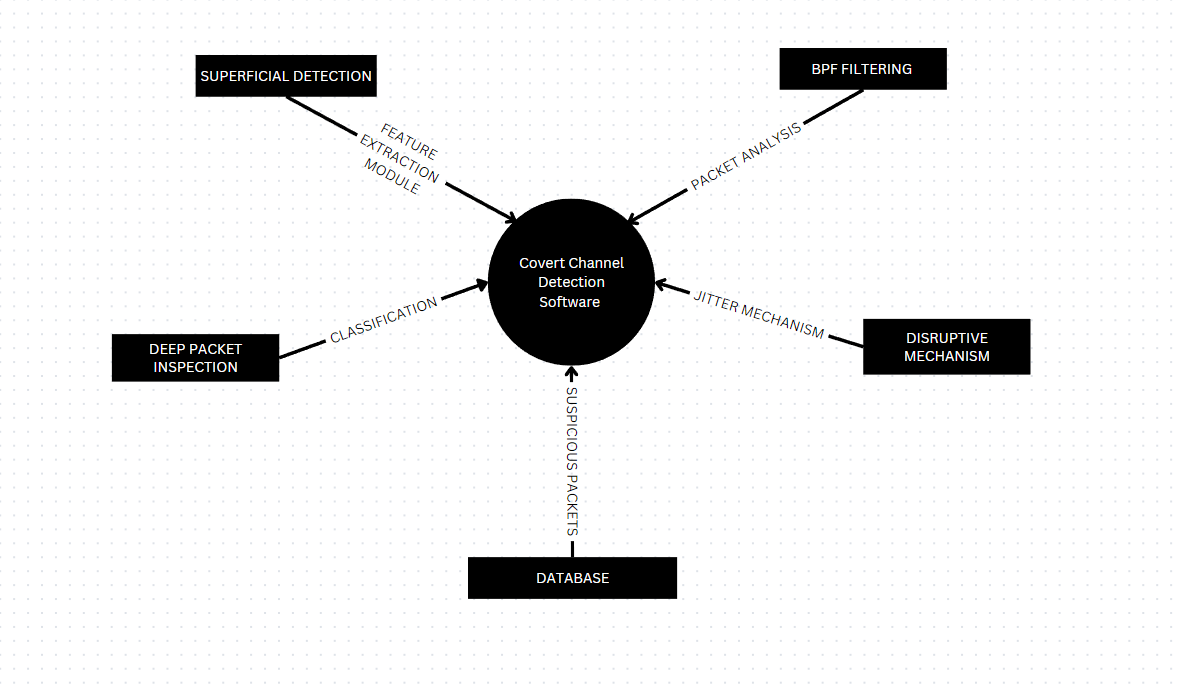


Figure 3: External Interfaces Diagram

**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**

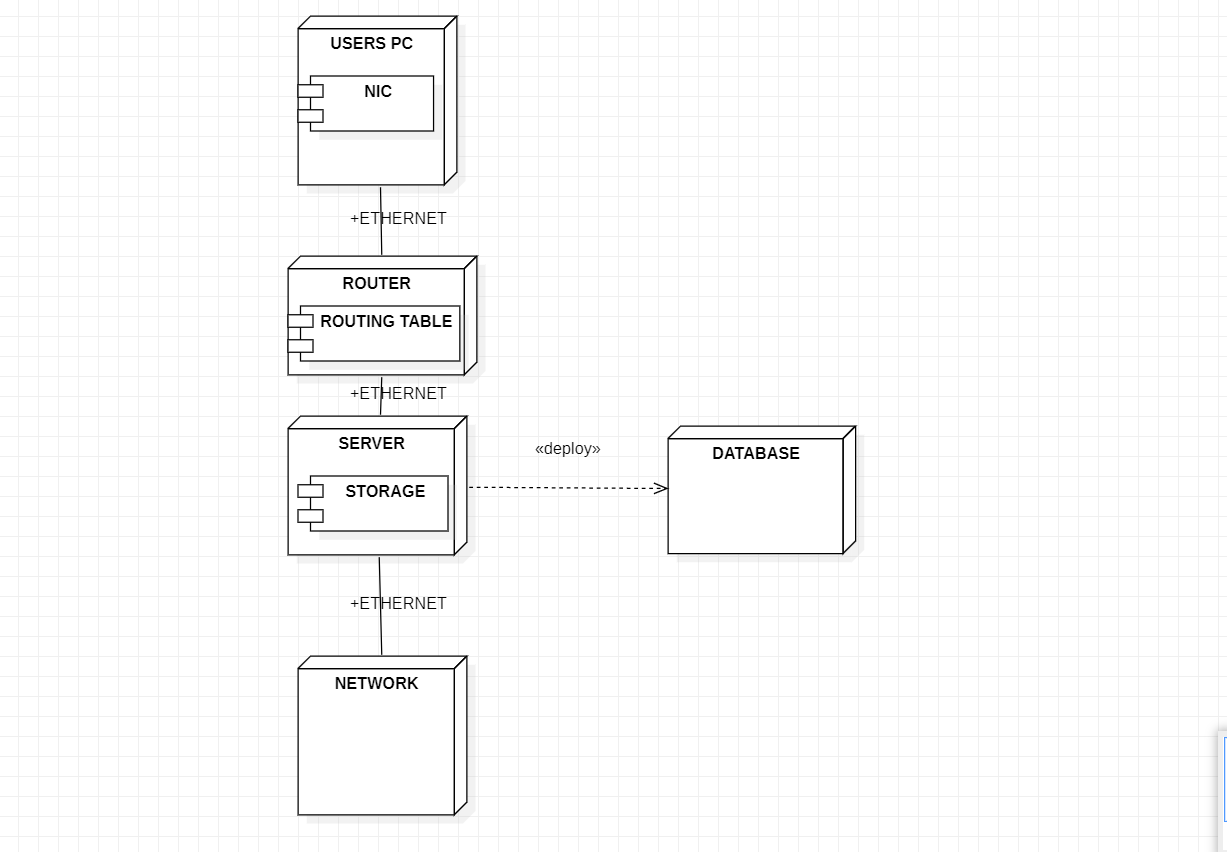


Figure 4: Packaging and Deployment Diagram

**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 Python.
* **Database Management System:** The system may depend on a specific database management system such as MySQL.
* **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.
5. Convert each small binary bitstring to integers between 0 and 15 say X.
6. 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 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**

* HLD - “High-Level Design”
* UML - “Unified Modelling 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**

* “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," “”
* "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. |  |  |  |  |