

A Project Report
On
Botnet Attack In Computer Network Security

Submitted by

M. Karthik

(20HQ1A4221)

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING - (AI & ML)

Under the Guidance of

Mrs. B. SAILAJA

(Assistant Professor)



AVANTHI'S RESEARCH & TECHNOLOGICAL ACADEMY

**(AFFILIATED TO JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY VIZIANAGARAM,
A.P)BASAVAPALEM, BHOGAPURAM(MANDAL), VIZIANAGARAM DISTRICT)**

AVANTHI'S RESEARCH AND TECHNOLOGICAL ACADEMY

(BASAVAPALEM, BHOGAPURAM MANDAL, VIZIANAGARAM DISTRICT)



BONAFIDE CERTIFICATE

Certified that this project report **“Botnet Attack In Computer Network Security”** is the bonified work **“ M.Sai Lakshman Karthik (20HQ1A4221)”** who carried out project work under our supervision. The results embodied in this project have been verified and found satisfactory.

INTERNAL GUIDE

Mrs. B. SAILAJA

Asst. Professor

Department of CSE

HEAD OF THE DEPARTMENT

Mrs. B. SAILAJA

Assistant Professor

Department of CSE-(AI&ML)

EXTERNAL EXAMINER

ACKNOWLEDGMENT

We would like to extend our sincere gratitude to all those who help us in our Project.

We express our sincere thanks to our guide **Mrs. B. SAILAJA** who has been source of inspiration for us throughout our project for her/his valuable advices in making our project success.

We owe our gratitude to our beloved **Mrs. B. SAILAJA**, HOD for assisting us to complete our project work.

We also thankful to our honorable principal sir **Dr. P. GOVINDA RAO** principal of **“AVANTHI’S RESEARCH AND TECHNOLOGICAL ACADEMY, BHOGHAPURAM”** who has shown keen interest in us and encouraged us by providing all the facilities to complete our project successfully.

We also thankful to our institution and our family members, without them this project would have been a distant reality.

Thanks, and appreciation from those who helped us and supported us to make our project successful.

PROJECT ASSOCIATES

M. Sai Lakshman Karthik

(20HQ1A4221)

DECLARATION

We **“M. Sai Lakshman Karthik (20HQ1A4221)”** hereby declare that the project report entitled **“Botnet Attack In Computer Network Security ”** is an original and authentic work done in the **Department of CSE-(AI&ML)**, by submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge.

PROJECT ASSOCIATES

M. Sai Lakshman Karthik

(20HQ1A4221)

ABSTRACT

Among the various forms of malware, Botnet is the serious threat which occurs commonly in today's cyber attacks and cyber crimes. Botnet are designed to perform predefined functions in an automated fashion, where these malicious activities ranges from online searching of data, accessing lists, moving files sharing channel information to DDoS attacks against critical targets, phishing, click fraud etc. Existence of command and control(C&C) infrastructure makes the functioning of Botnet unique; in turn throws challenges in the mitigation of Botnet attacks. Hence Botnet detection has been an interesting research topic related to cyber-threat and cyber-crime prevention in network security. Various types of techniques and approaches have been proposed for detection, mitigation and prevention to Botnet attack. Here I discusses in detail about Botnet and related research including Botnet evolution, life-cycle, command and control models, communication protocols, Botnet detection, and Botnet mitigation mechanism etc. Also an overview of research on Botnets which describe the possible attacks performed by various types of Botnet communication technologies in future.

KEYWORDS— Bot; Botnet; C&C mechanism; communication protocols; honeynet; passive traffic; attacks; defense; prevention; mitigation

LIST OF FIGURES

Table of Contents

CHAPTER-1	9
INTRODUCTION	9
INTRODUCTION	10
1.1: LITERATURE SURVEY:.....	15
CHAPTER-2	17
SYSTEM ANALYSIS.....	17
2.1. EXISTING SYSTEM:	18
2.2. PROPOSED SYSTEM:	19
2.3. SYSTEM REQUIREMENTS:.....	20
2.4. SOFTWARE ENVIRONMENT:	20
2.4.1. The Java Standard Library:	22
2.4.2. Dealing with Bugs:	23
2.4.3. Documentation Bugs:.....	25
2.4.4. Using the Java Issue Tracker:	27
2.5. FEASIBILITY STUDY:	29
CHAPTER-3	32
ARCHITECTURE	32
ARCHITECTURE	33
WORKING OF ALGORITHM	34
CHAPTER-4	36
SYSTEM DESIGN	36
UML DIAGRAM.....	37
DATA FLOW DIAGRAM.....	43
CHAPTER-5	45
CODING	45
SOURCE CODE.....	46
CHAPTER-6	84
OUTPUTS.....	84
CHAPTER-7	90
TESTING.....	90
SYSTEM TESTING	91
TYPES OF TESTS	91
Unit Testing.....	91

Integration Testing	91
Functional Test.....	92
System Test	92
White Box Testing	92
Black Box Testing.....	93
CHAPTER-8	94
CONCLUSION.....	94
CONCLUSION.....	95
CHAPTER-9	96
FUTURE SCOPE.....	96
FUTURE SCOPE.....	97
CHAPTER-10	98
REFERENCES	98
REFERENCES	99

CHAPTER-1

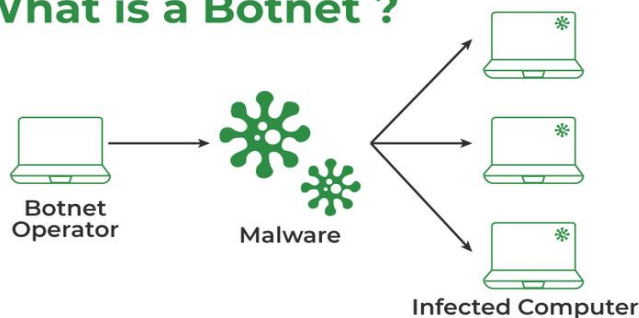
INTRODUCTION

INTRODUCTION

A Network of compromised computers is called a botnet. Compromised computers are also called Zombies or Bots. This software is mostly written in C++, C, Java, Python. The main motive of botnet is that it starts with the dark side of the internet which introduced a new kind of Crime called Cybercrime.

Among the malware (malicious software) botnet is the most widespread and severe threat. Several large institutions, government organizations, and almost every social networking website Facebook, Twitter, Instagram, etc, e-commerce website Amazon, and Flipkart, etc, in short, every firm associated with the internet became the victim of this malware. This kind of malicious software is freely available in the market for lease. It can be used in DDoS attacks (Smurf Attacks), Phishing, Extortion, etc.

What is a Botnet ?



Botnets

Botnet Communication:

At first, those who want to be botmaster finds the target system (here target system means finding the vulnerable system), then use popular social engineering techniques like phishing, click fraud, etc to install a small (Kbs) executable file into it. A small patch has been included in the code, making it not visible even with the running background process. A naive user won't even come to know that his/her system became part of a bot army. After infection, the bot looks for the channel through which it can communicate with its master. Mostly Channel (command and Control channel) uses the existing protocol to request the command and receive updates from the master, so if anyone tries to look at the traffic behavior then it will be quite

difficult to figure it out. Botmaster is used to write scripts to run an executable file on different OS.

The following are the major things that can be performed on bots:

- Web-Injection: **Botmaster can inject snippets of code to any secured website that which bot used to visit.**
- Web filters: **Here on use a special symbol like:”!” for bypassing a specific domain, and “@” for the screenshot used.**
- Web-fakes: **Redirection of the webpage can be done here.**
- DnsMAP: **Assign any IP to any domain which the master wants to route to the bot family.**

Types of Botnet:

Here are the types of botnets mentioned below based on the Channel.

Internet Relay Chat (IRC) Botnet:

Internet Relay Chat (IRC) acts as the C&C Channel. Bots receive commands from a centralized IRC server. A command is in the form of a normal chat message. The limitation of the Internet Relay Chat (IRC) Botnet is that the Entire botnet can be collapsed by simply shutting down the IRC Server.

Peer-to-Peer (P2P) Botnet:

It is formed using the P2P protocols and a decentralized network of nodes. Very difficult to shut down due to its decentralized structure. Each P2P bot can act both as the client and the server. The bots frequently communicate with each other and send “keep alive” messages. The limitation of Peer-to-Peer Botnets is that it has a higher latency for data transmission.

Hyper Text Transfer Protocol (HTTP) Botnet:

Centralized structure, using HTTP protocol to hide their activities. Bots use specific URLs or IP addresses to connect to the C&C Server, at regular intervals. Unlike IRC bots, HTTP bots periodically visit the C&C server to get updates or new commands.

How Does it Work?

The working of the Botnet can be defined as either you writing code to build software or using it from the available (Leaked) botnet like ZEUS Botnet (king of all botnet), Mirai botnet, BASHLITE, etc. then finding the vulnerable system where you can install this software through some means like social engineering (e.g Phishing) soon that system becomes a part of a bot army. Those who control it are called the botmaster which communicates its bot army using a command and control channel.

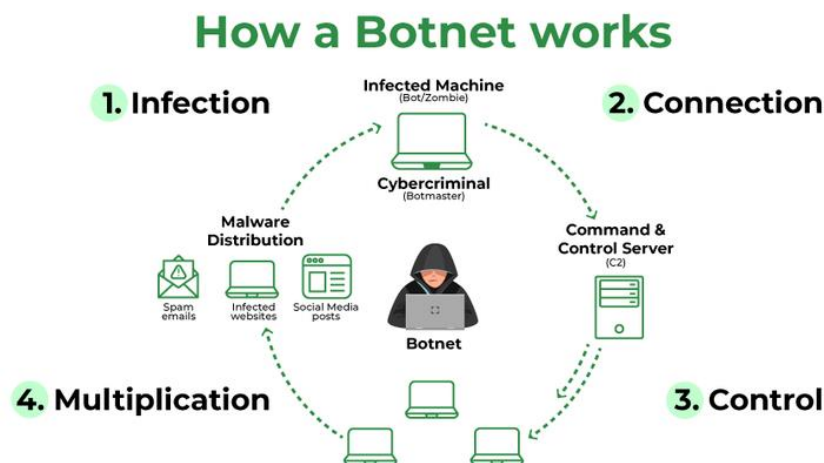


Fig : 1.1

Types of Botnet Attacks:

Below mentioned are the attacks performed by the Botnets.

- **Phishing:** Botnets help in distributing malware and suspicious activities via Phishing emails. These include a multiple number of bots and the whole process is automated and it is difficult to shut down.
- **Distributed Denial-of-Service (DDoS) Attack:** DDoS Attack is a type of attack performed by the Botnets in which multiple requests are sent that leads to the crash of a particular application or server. DDoS Attacks by Network Layer use SYN

Floods, UDP Floods, etc to grasp the target's bandwidth and let them protect from being attacked.

- **Spambots:** Spambots are a type of Botnet Attack, where they take emails from websites, guestbooks, or anywhere an email id is required to log in. This section covers more than 80 percent of spam.
- **Targeted Intrusion:** This is one of the most dangerous attacks as they attack the most valuable thing or data, valuable property, etc.

How to Protect Against Botnets?

- The most important way to protect from Botnets is to give training to users about identifying suspicious links.
- Keep the system software always updated to become safe from the Botnets.
- Using two-factor authentication is a way to be safe from the Botnet.
- There are several antiviruses present in the market which keeps you protected from Botnets.
- Try to change passwords on a regular basis for better protection from Botnets.

Botnet Lifecycle:

Botnet Lifecycle can be understood with the help of the following diagrams. Here we have illustrated the lifecycle of Botnet in 4 stages as shown in the figure.

Stage-1:

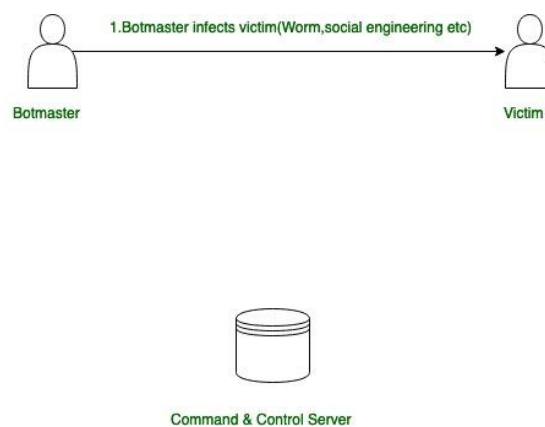


Fig : 1.

Stage-2:

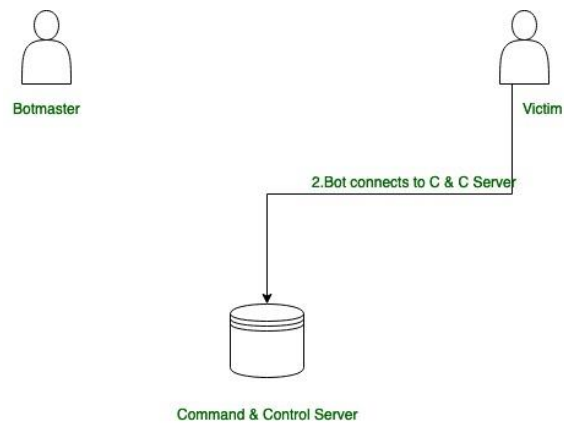


Fig : 1.3

Stage-3:

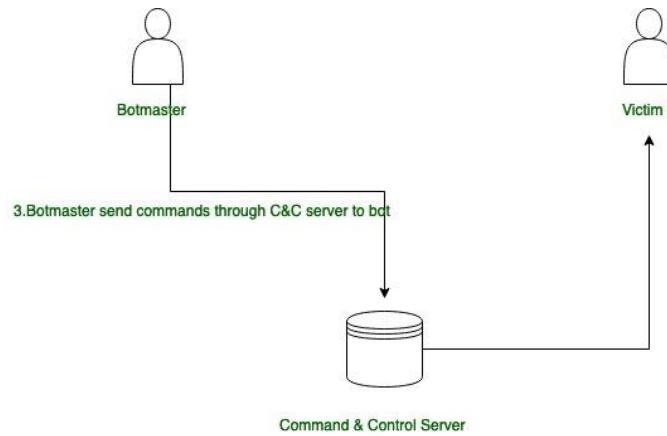


Fig : 1.4

Stage-4:

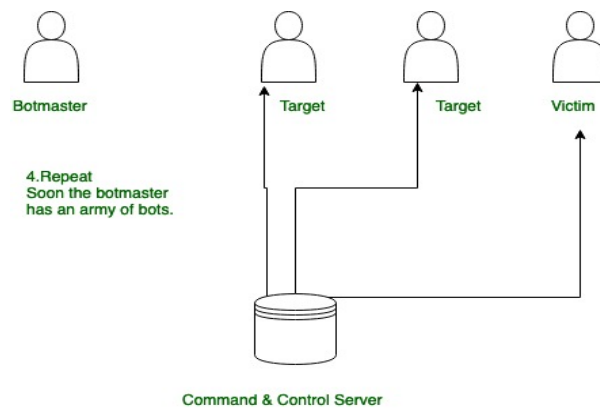


Fig : 1.5

1.1: LITERATURE SURVEY:

A literature survey on botnet attacks in computer network security would involve exploring a wide range of research articles, academic papers, conference proceedings, and technical reports. Here's a structured approach you might consider for conducting such a survey:

- 1. Define Scope and Objectives:** Clearly define the scope of your literature survey. Are you focusing on specific types of botnet attacks, such as DDoS attacks, spamming, information theft, or others? What aspects of computer network security are you interested in, such as detection, prevention, mitigation, or forensic analysis?
- 2. Search Strategy:** Identify relevant keywords and phrases related to botnet attacks and computer network security. Utilize academic databases (such as IEEE Xplore, ACM Digital Library, Google Scholar), search engines, and library catalogs to find relevant literature. Boolean operators and advanced search techniques can help refine your search results.
- 3. Literature Selection:** Evaluate the search results to select relevant literature for your survey. Consider factors such as publication date, relevance to your research objectives, credibility of the sources (peer-reviewed journals, reputable conferences), and the significance of the contributions.
- 4. Review and Summarize:** Read the selected literature thoroughly and summarize the key findings, methodologies, approaches, and insights related to botnet attacks in computer network security. Identify common themes, trends, and challenges addressed by researchers in the field.
- 5. Classification and Categorization:** Organize the literature into categories based on different aspects of botnet attacks, such as attack vectors, detection techniques, mitigation

strategies, case studies, and experimental evaluations. This classification will help you structure your survey and present the information effectively.

- 6. Critical Analysis:** Evaluate the strengths and weaknesses of the existing approaches and methodologies discussed in the literature. Identify gaps in the current research and areas that require further investigation or improvement.
- 7. Synthesis and Conclusion:** Synthesize the findings from the literature survey to provide a comprehensive overview of the state-of-the-art in botnet attacks and computer network security. Highlight emerging trends, unresolved issues, and future research directions. Conclude with insights into the implications of the existing literature for addressing real-world challenges in securing computer networks against botnet threats.
- 8. References and Citations:** Ensure proper citation of the sources used in your literature survey following the appropriate citation style (e.g., APA, MLA, IEEE). Provide a comprehensive list of references to acknowledge the contributions of previous research and enable readers to explore the literature further.

By following these steps, you can conduct a thorough literature survey on botnet attacks in computer network security, providing valuable insights into the current state of research and guiding future studies in the field.

CHAPTER-2

SYSTEM ANALYSIS

2.1. EXISTING SYSTEM:

The current state of computer network security faces significant challenges in detecting and mitigating botnet attacks. Traditional security systems may lack the sophistication required to identify the intricate and dynamic nature of botnet activities, leading to potential security breaches and data compromises.

DISADVANTAGES:

1. The scheme relies on the presence of cryptographic vulnerabilities or client-side zero-day vulnerabilities that can be exploited to execute the Pebbleware code at the botmaster's machine.
2. If the botmaster only communicates with the victims and does not retrieve any stolen data, more intelligence may need to be integrated into the Pebbleware for tracing.
3. The paper focuses only on identifying symmetric encryption keys. Identifying asymmetric keys is mentioned as a challenging open problem.
4. Defending against botnets leveraging anonymous networks and social networks in cloud environments is stated as an intriguing future research direction.

2.2. PROPOSED SYSTEM:

The proposed system presents an advanced approach to combat botnet attacks through the integration of Suggested Key Identification Scheme, which is a type of Botnet Signature Matching Algorithm. By leveraging Python and web technologies, the project introduces a responsive and dynamic system capable of identifying, isolating, and neutralising botnets to safeguard the integrity and availability of network resources through Pebbleware and Traceback Servers.

ADVANTAGES:

1. It can identify the encryption keys used in botnet communications without having access to the source code or meaningful plaintext, which is a major challenge. This allows decrypting the botnet traffic.
2. It proposes a method to piggyback specially designed executable code called "Pebbleware" on the botnet communication packets. When executed at the botmaster's machine, Pebbleware reports back the host information, enabling tracing back to the botmaster through stepping stones like proxies, VPNs etc.
3. The key identification scheme has low false positives, avoiding alerting the attacker about the traceback attempt.
4. It is efficient in narrowing down candidate encryption keys from the memory images through a multi-phase process.

2.3. SYSTEM REQUIREMENTS:

HARDWARE REQUIREMENTS:

System: Laptop & Computer

Processor: i5 CORE

Storage: 128gb and Above

Ram: 4gb and Above

OS: Linux, Windows 10 & Above

SOFTWARE REQUIREMENTS:

Software: Java-8u20 Version

Database: Mysql, Mango DB, Oracle

Servers: Mysql, Xampp

Coding Language: Java & Sql

Drivers: JDBC, ODBC

2.4. SOFTWARE ENVIRONMENT:

Java is a versatile and robust programming language that has been a cornerstone of software development for over two decades. Known for its portability, reliability, and scalability, Java has become a staple in a wide range of applications, from enterprise systems to mobile apps and web development.

One of Java's key strengths is its "write once, run anywhere" mantra, facilitated by the Java Virtual Machine (JVM). Java code is compiled into bytecode, which can run on any device or platform that has a compatible JVM installed. This platform independence has made Java

an ideal choice for developing cross-platform applications, minimizing the need for code modifications across different environments.

Java's object-oriented programming (OOP) paradigm promotes code reusability, modularity, and maintainability. Classes and objects form the building blocks of Java programs, allowing developers to encapsulate data and behavior within cohesive units. Inheritance, polymorphism, and encapsulation are fundamental concepts in Java OOP, empowering developers to create scalable and extensible software solutions.

The Java Standard Edition (Java SE) provides core libraries and APIs for general-purpose programming tasks, including I/O operations, networking, concurrency, and data manipulation. These built-in functionalities streamline development and enable developers to focus on solving business problems rather than reinventing the wheel.

Java's extensive ecosystem of third-party libraries and frameworks further enhances its capabilities and productivity. Frameworks like Spring, Hibernate, and Apache Struts facilitate the development of enterprise-grade applications, providing solutions for dependency injection, persistence, and web application development.

In addition to its server-side prowess, Java has also made significant strides in the mobile development space with the introduction of Android. Android Studio, the official IDE for Android development, leverages Java's power and flexibility to create feature-rich mobile applications for a diverse range of devices.

Moreover, Java's community-driven approach fosters continuous innovation and improvement. The Java Community Process (JCP) enables developers, organizations, and stakeholders to contribute to the evolution of the Java platform through the submission and review of Java Specification Requests (JSRs).

Overall, Java's combination of portability, performance, and community support makes it a compelling choice for developers seeking to build robust, scalable, and maintainable software solutions across diverse domains and platforms.

2.4.1. The Java Standard Library:

The Java Standard Library, also known as the Java Standard Edition (Java SE) API, is a comprehensive collection of classes, interfaces, and packages provided by Oracle as part of the Java Development Kit (JDK). It forms the foundation of Java programming and provides essential functionalities for a wide range of tasks, including input/output operations, networking, concurrency, data manipulation, and more. Here are some key components of the Java Standard Library:

1. java.lang: This package contains fundamental classes and interfaces that are automatically imported into every Java program. It includes classes like `Object`, `String`, and `System`, as well as essential interfaces such as `Comparable` and `Runnable`.

2. java.util: The `java.util` package provides a variety of utility classes and data structures for common programming tasks. It includes classes for collections (e.g., `ArrayList`, `HashMap`, `LinkedList`), date and time handling (`Date`, `Calendar`, `DateFormat`), and utility functions (`Arrays`, `Collections`, `Random`).

3. java.io: The `java.io` package offers classes for performing input and output operations, such as reading from and writing to files, streams, and other I/O sources. Key classes include `File`, `InputStream`, `OutputStream`, `Reader`, and `Writer`.

4. java.net: This package contains classes for networking operations, enabling Java programs to communicate over the network. It includes classes like `Socket`, `ServerSocket`, `URL`, and `URLConnection` for creating client-server applications, accessing web resources, and implementing network protocols.

5. java.concurrent: The `java.concurrent` package provides classes and interfaces for concurrent and multithreaded programming. It includes utilities for thread management (`Thread`, `Executor`, `ExecutorService`), synchronization (`Lock`, `Semaphore`, `Condition`), and concurrent data structures (`ConcurrentHashMap`, `BlockingQueue`).

6. java.math: The `java.math` package offers classes for arbitrary-precision arithmetic and mathematical operations. It includes classes like `BigInteger` and `BigDecimal`, which support operations on integers and floating-point numbers with arbitrary precision.

7. java.security: This package provides classes and interfaces for implementing security-related functionalities, such as cryptography, secure random number generation, digital signatures, and access control. Key classes include `MessageDigest`, `Cipher`, and `KeyStore`.

8. java.awt and javax.swing: These packages contain classes for creating graphical user interfaces (GUIs) in Java. `java.awt` provides basic GUI components and graphics rendering capabilities, while `javax.swing` offers a more extensive set of components and support for features like event handling, layout management, and look-and-feel customization.

These are just a few examples of the many packages and classes available in the Java Standard Library. Together, they provide a robust and versatile toolkit for Java developers to build a wide variety of applications efficiently and effectively.

2.4.2. Dealing with Bugs:

Dealing with bugs in Java involves a systematic approach to identifying, diagnosing, and fixing issues in your code. Here's a step-by-step guide to effectively handle bugs in Java:

1. Reproduce the Bug: Before you can fix a bug, you need to reproduce it consistently. Start by understanding the conditions and inputs that trigger the bug. Create test cases or scenarios that demonstrate the issue reliably.

2. Isolate the Problem: Once you've reproduced the bug, isolate the problematic code or component. Use debugging techniques such as print statements, logging, or debugging tools provided by your Integrated Development Environment (IDE) to narrow down the scope of the issue.

3. Understand the Code: Take the time to understand the code surrounding the bug. Review relevant documentation, comments, and design decisions to gain insights into the intended behavior and potential causes of the bug.

4. Analyze Error Messages: Pay attention to any error messages, exceptions, or stack traces associated with the bug. They often provide valuable clues about the nature and location of the problem.

5. Use Debugging Tools: Leverage the debugging features provided by your IDE, such as breakpoints, step-by-step execution, variable inspection, and expression evaluation. Debugging tools help you examine the program's state and execution flow in real-time, making it easier to identify the root cause of the bug.

6. Write Unit Tests: Create unit tests to validate the expected behavior of the code and prevent regressions. Test-driven development (TDD) practices can help catch bugs early in the development process and ensure code correctness over time.

7. Consult Documentation and Community Resources: If you encounter unfamiliar behavior or errors, refer to the official Java documentation, API references, and community

forums. Online communities like Stack Overflow often provide valuable insights, solutions, and workarounds for common Java bugs and issues.

8. Version Control and Code Reviews: If you're working in a team, use version control systems like Git to track changes and collaborate on bug fixes. Conduct code reviews to leverage the collective expertise of your team members and ensure code quality.

9. Apply Systematic Debugging Techniques: If the bug persists despite your efforts, consider using systematic debugging techniques such as binary search debugging, divide-and-conquer, or hypothesis-driven debugging to systematically eliminate potential causes and narrow down the problem.

10. Implement Fixes and Regression Testing: Once you've identified the root cause of the bug, implement the necessary code changes or fixes. Be sure to test your fixes thoroughly using regression testing to verify that they resolve the issue without introducing new bugs or regressions.

11. Document and Learn from the Experience: Document the bug, its root cause, and the steps taken to fix it. Share your findings with your team members and update relevant documentation or knowledge repositories. Use the experience as an opportunity to learn and improve your coding practices to prevent similar bugs in the future.

By following these steps and adopting a systematic approach to bug fixing, you can effectively identify and resolve bugs in your Java codebase, improving its reliability, maintainability, and overall quality.

2.4.3. Documentation Bugs:

Documenting bugs in Java is a crucial aspect of software development that facilitates effective communication, collaboration, and problem-solving among developers, testers, and

other stakeholders. Proper documentation of bugs ensures that issues are accurately recorded, tracked, and resolved in a timely manner. Here are some key points to consider when documenting bugs in Java:

1. **Clear Description:** Start by providing a clear and concise description of the bug, including details such as the symptoms observed, expected behavior, and steps to reproduce the issue. Use descriptive language that accurately conveys the nature and impact of the bug to ensure that developers and testers understand the problem.
2. **Reproducibility:** Document the steps or conditions required to reproduce the bug reliably. Include information about the specific environment, inputs, configurations, and dependencies necessary to trigger the issue. Reproducible bugs are easier to diagnose and fix, leading to faster resolution.
3. **Severity and Priority:** Assign appropriate severity and priority levels to the bug based on its impact on the software functionality, user experience, and project timelines. Classify bugs as critical, major, minor, or cosmetic to prioritize them effectively and allocate resources accordingly.
4. **Code Samples and Stack Traces:** If applicable, include code snippets, stack traces, or error messages associated with the bug. These artifacts provide valuable insights into the root cause of the issue and help developers pinpoint the problematic code or component more efficiently.
5. **Attachments and Screenshots:** Attach relevant files, logs, screenshots, or test cases to provide additional context and evidence of the bug. Visual aids can help illustrate the problem visually and assist developers in understanding the issue better.
6. **Version Information:** Specify the version of the software or libraries affected by the bug, as well as the platform, operating system, and environment where the issue was observed. Version information helps developers identify potential compatibility issues and track bug fixes across different releases.
7. **Steps to Reproduce:** Document the exact steps required to reproduce the bug, including any specific user actions, inputs, or configurations. Provide a clear, step-by-step guide that enables developers and testers to replicate the issue consistently and validate potential fixes.
8. **Collaboration and Comments:** Encourage collaboration and feedback by allowing team members to add comments, suggestions, or updates to the bug report. Foster open

communication channels where stakeholders can share insights, propose solutions, and track the progress of bug resolution efforts.

- 9. Status and Resolution:** Track the status of the bug throughout its lifecycle, from initial report to resolution. Use status labels such as "open," "assigned," "in progress," "resolved," and "closed" to indicate the current state of the bug and its resolution status.
- 10. Regression Testing:** After the bug is fixed, perform regression testing to verify that the issue has been resolved satisfactorily and that no new bugs or regressions have been introduced. Update the bug report with the results of regression testing and confirm the closure of the bug.

By documenting bugs effectively in Java, teams can streamline the bug tracking and resolution process, foster collaboration, and ultimately deliver higher-quality software products to end users.

2.4.4. Using the Java Issue Tracker:

Using a Java issue tracker is essential for efficiently managing and tracking bugs, feature requests, and other tasks throughout the software development lifecycle. One popular issue tracker used in Java development is JIRA, developed by Atlassian. Here's how you can utilize JIRA for effective issue tracking in Java projects:

- 1. Creating Issues:** Start by creating issues in JIRA to represent bugs, new features, improvements, or other tasks. Provide detailed descriptions, including steps to reproduce, expected behavior, actual behavior, and any relevant attachments or screenshots.
- 2. Assigning and Prioritizing:** Assign each issue to the appropriate team member responsible for addressing it. Prioritize issues based on severity, impact, and urgency to ensure that critical bugs and high-priority tasks are addressed promptly.
- 3. Organizing Issues:** Use JIRA's flexible project and issue organization features to categorize and manage issues effectively. Create custom workflows, issue types, and statuses tailored to your project's needs, such as "To Do," "In Progress," "Code Review," and "Resolved."

4. Tracking Progress: Monitor the progress of issues as they move through various stages of resolution. Update issue statuses, add comments, and log work to provide visibility into the status of each issue and facilitate collaboration among team members.

5. Linking and Dependencies: Establish relationships between issues by linking related tasks, dependencies, or blocking relationships. Use JIRA's linking features to track dependencies between issues and ensure that tasks are completed in the correct sequence.

6. Integration with Version Control: Integrate JIRA with version control systems like Git to streamline issue tracking and development workflows. Link commits, branches, and pull requests to JIRA issues to provide traceability between code changes and corresponding tasks.

7. Automated Notifications: Configure JIRA to send automated notifications and alerts to team members when issues are assigned, updated, or resolved. Keep stakeholders informed about the progress of tasks and ensure timely communication within the team.

8. Reporting and Analytics: Utilize JIRA's reporting and analytics features to generate insights into project performance, issue trends, and team productivity. Create custom dashboards, charts, and metrics to track key performance indicators and identify areas for improvement.

9. Customization and Extensions: Customize JIRA to align with your team's workflows, processes, and preferences. Explore available plugins, extensions, and integrations to enhance JIRA's functionality and adapt it to your project's specific requirements.

10. Continuous Improvement: Regularly review and refine your use of JIRA to optimize issue tracking and project management processes. Solicit feedback from team members, identify pain points, and implement changes to improve efficiency, transparency, and collaboration.

By leveraging JIRA's powerful features and capabilities, Java development teams can streamline issue tracking, improve collaboration, and deliver high-quality software products more effectively.

2.5. FEASIBILITY STUDY:

A feasibility study of a botnet attack in computer network security (CNS) involves assessing the practicality, risks, and potential impacts of orchestrating such an attack. Here's how you might approach conducting a feasibility study for a botnet attack in CNS:

1. Understanding Botnets: Begin by gaining a comprehensive understanding of what botnets are, how they operate, and their typical attack vectors. Botnets are networks of compromised devices (bots) controlled by a central command and control (C&C) server. They are often used for malicious activities such as distributed denial of service (DDoS) attacks, spamming, information theft, and propagation of malware.

2. Assessing Attack Objectives: Define the specific objectives of the botnet attack in CNS. Are you aiming to disrupt network operations, steal sensitive information, gain unauthorized access to systems, or achieve other malicious goals? Understanding the attacker's objectives is crucial for evaluating the feasibility and potential impacts of the attack.

3. Identifying Target Systems: Determine the target systems or assets within the CNS that the botnet attack aims to compromise or exploit. This could include servers, workstations, routers, IoT devices, or other network-connected devices. Assess the security vulnerabilities, attack surface, and potential entry points within the target network.

4. Analyzing Attack Techniques: Evaluate the various attack techniques and strategies that could be employed by the botnet to compromise target systems and evade detection. This may include exploiting software vulnerabilities, leveraging social engineering tactics, conducting phishing attacks, or using malware propagation techniques.

5. Assessing Detection and Mitigation: Consider the existing detection and mitigation measures in place within the CNS to identify and thwart botnet attacks. Evaluate the effectiveness of intrusion detection systems (IDS), firewalls, antivirus software, network traffic analysis tools, and other security controls in detecting and mitigating botnet activity.

6. Evaluating Legal and Ethical Considerations: Consider the legal and ethical implications of conducting a botnet attack in CNS. Botnet attacks are illegal and may result in severe legal consequences, including criminal prosecution and civil penalties. Additionally, consider the ethical implications of causing harm or disruption to network infrastructure, data, and users.

7. Assessing Impact and Consequences: Evaluate the potential impact and consequences of a botnet attack on the targeted CNS. Consider factors such as downtime, data loss, financial losses, reputational damage, regulatory compliance violations, and operational disruptions. Assess the likelihood and severity of these impacts based on the attack scenario and the resilience of the target network.

8. Risk Analysis and Mitigation: Conduct a risk analysis to identify and prioritize potential risks associated with the botnet attack in CNS. Develop mitigation strategies and countermeasures to reduce the likelihood and impact of these risks. This may include implementing security patches, network segmentation, access controls, incident response procedures, and employee training.

9. Cost-Benefit Analysis: Perform a cost-benefit analysis to weigh the potential benefits of conducting the botnet attack against the associated costs, risks, and consequences. Consider factors such as the resources required to execute the attack, the potential gains from successful exploitation, and the potential losses from detection and countermeasures.

10. Documentation and Reporting: Document the findings of the feasibility study, including the assessment of attack objectives, target systems, attack techniques, detection and mitigation

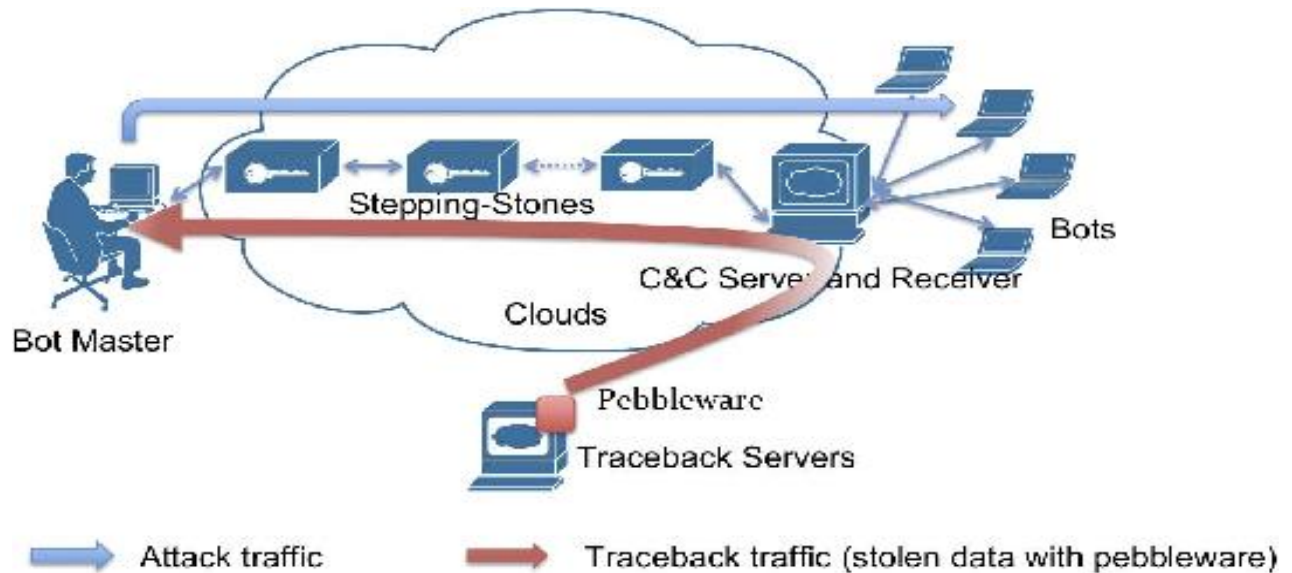
measures, legal and ethical considerations, impact analysis, risk assessment, and cost-benefit analysis. Present the findings in a comprehensive report to stakeholders, highlighting key insights and recommendations for decision-making.

By conducting a thorough feasibility study of a botnet attack in CNS, organizations can better understand the risks and potential impacts of such attacks, and develop proactive strategies to enhance network security and resilience against emerging threats.

CHAPTER-3

ARCHITECTURE

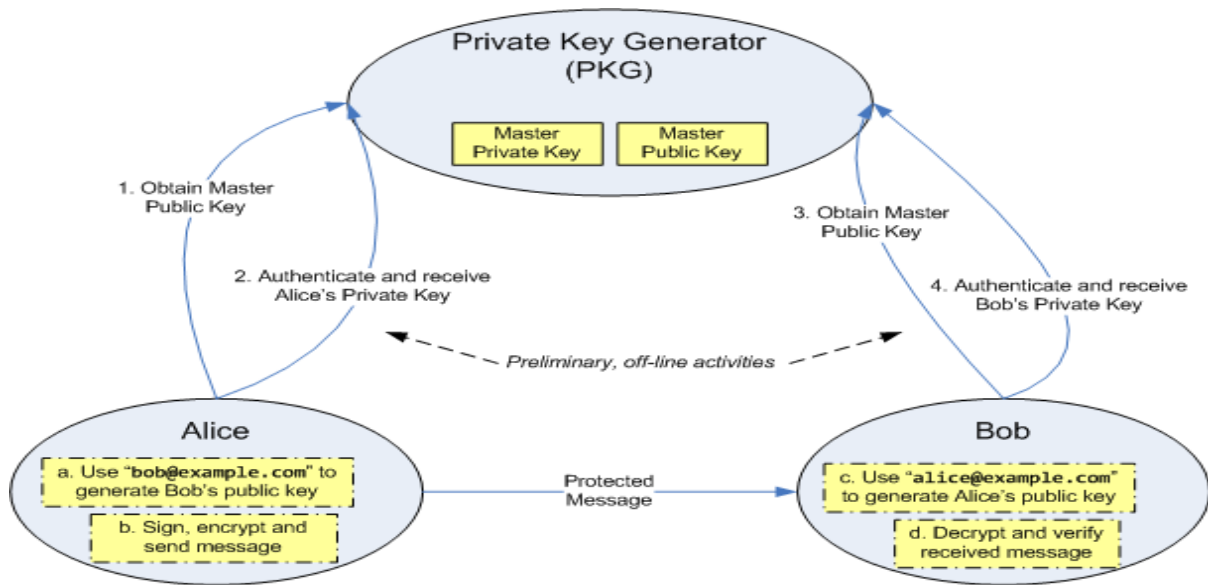
ARCHITECTURE



- **BOTMASTER:-** A botmaster in a botnet attack in Computer Network Security (CNS) The botmaster is responsible for orchestrating the development and administration of the botnet, recruiting and compromising devices, establishing control through a Command and Control (C&C) server, issuing commands to the compromised devices, maintaining the botnet, and utilizing it to carry out various activities.
- **PEBBLEWARE:-** Once executed on the botmaster's machine, Pebbleware sends the host machine's IP address to a traceback server, allowing for the identification and potential takedown of botnets through a series of strategic steps.
- **C&C SERVERS:-** Command and Control (C&C) servers play a pivotal role in botnet are crucial for maintaining control over compromised machines, coordinating attacks, and serving as the headquarters for the botnet.
- **TRACEBACK SERVERS:-** Traceback servers play a crucial role in tracking the source of malicious activities in the context of botnets and cyberattacks. These servers are designed to trace back the path from a victim PC to a bot, and from the bot to a Command and Control (C&C) server. By utilizing host-based traceback schemes, traceback servers can help identify the origins of cyber threats.

WORKING OF ALGORITHM

BOTNET SIGNATURE MATCHING ALGORITHM-(Suggested Key Identification Scheme)



- **PRIVATE KEY GENERATOR:-** The private key generator tools available online allow users to create private keys for encryption purposes. These tools enable the generation of private keys of varying lengths, such as 1024-bit or 4096-bit keys, which are essential for secure communication and data protection. By using these generators, users can create private keys and corresponding public keys, facilitating secure encryption and decryption processes.
- **MASTER PRIVATE KEY:-** The master private key in botnet attacks serves as a fundamental component used in hierarchical deterministic wallets to derive all public and private key pairs from a single master seed value. In the context of botnets, the master private key plays a crucial role in enabling the generation of key pairs that are used to control the botnet's operations securely. By utilizing the master private key, botmasters can derive all necessary private keys to manage the addresses associated with the botnet.
- **MASTER PUBLIC KEY:-** The purpose of the master public key in botnet attacks is to facilitate the secure and deterministic generation of key pairs within the botnet ecosystem. The master public key, derived from the master private key, plays a crucial role in managing the different addresses associated with the botnet. These public keys allow for the secure sharing of information and commands among the compromised

devices, enabling the botmaster to maintain control over the botnet and orchestrate malicious activities effectively.

CHAPTER-4

SYSTEM DESIGN

UML DIAGRAM

UML (Unified Modelling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. It was initially started to capture the behaviour of complex software and non-software system and now it has become an OMG standard. This tutorial gives a complete understanding on UML.

Goals:

The Primary goals in the design of the UML are as follows:

- 1) Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- 2) Provide extendibility and specialization mechanisms to extend the core concepts.
- 3) Be independent of particular programming languages and development process.
- 4) Provide a formal basis for understanding the modeling language.
- 5) Encourage the growth of OO tools market.
- 6) Support higher level development concepts such as collaborations, frameworks, patterns and components.
- 7) Integrate best practices

4.1. UML USE DAIGRAM :

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to how what system functions are performed for which actor. Roles of the actors in the system can be depicted.

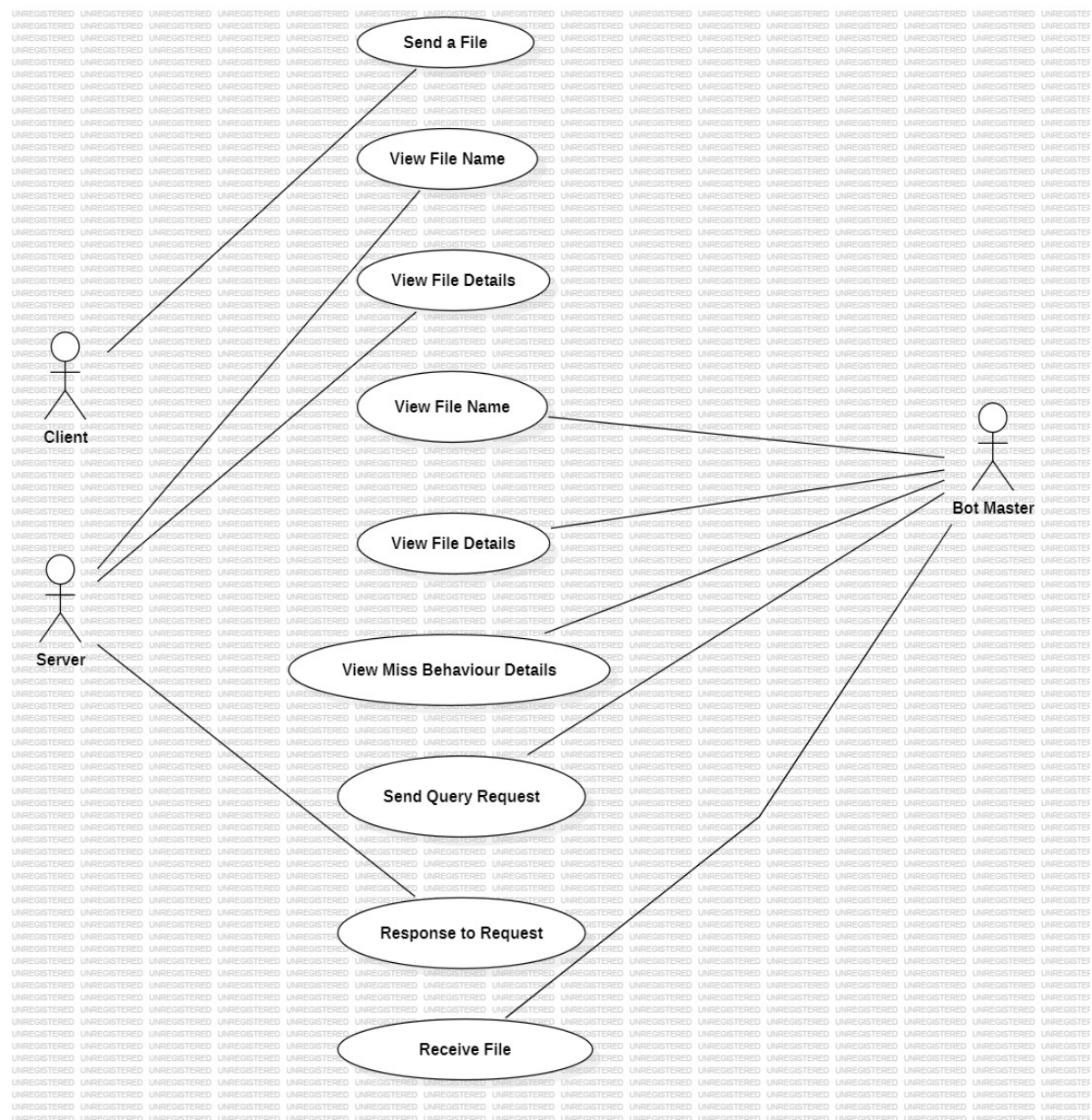


Fig 4.1. Use case diagram

4.2. SEQUENCE DIAGRAM :

A sequence diagram in Unified Modelling Language(UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

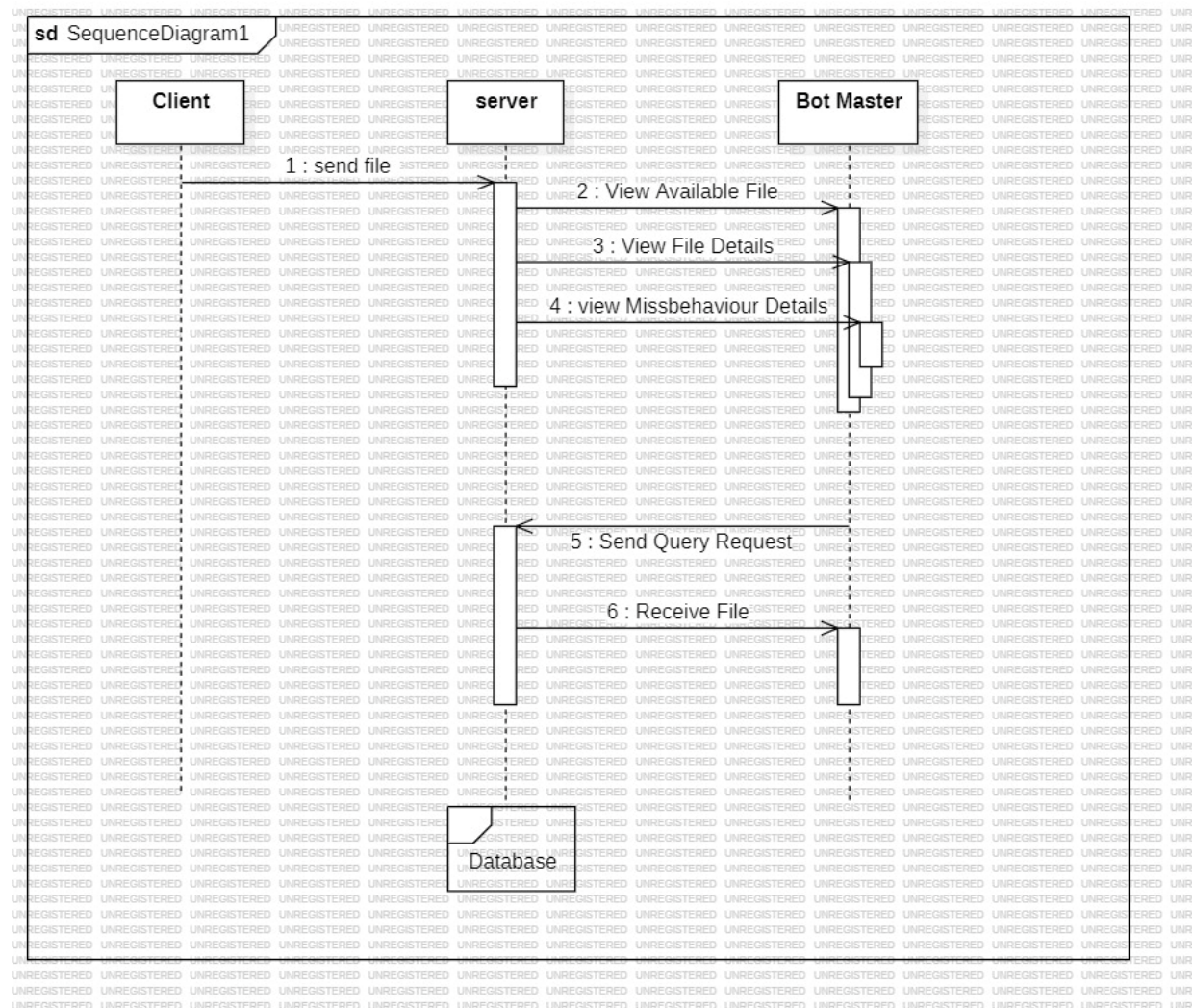


Fig 4.2. Sequence Diagram

4.3 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

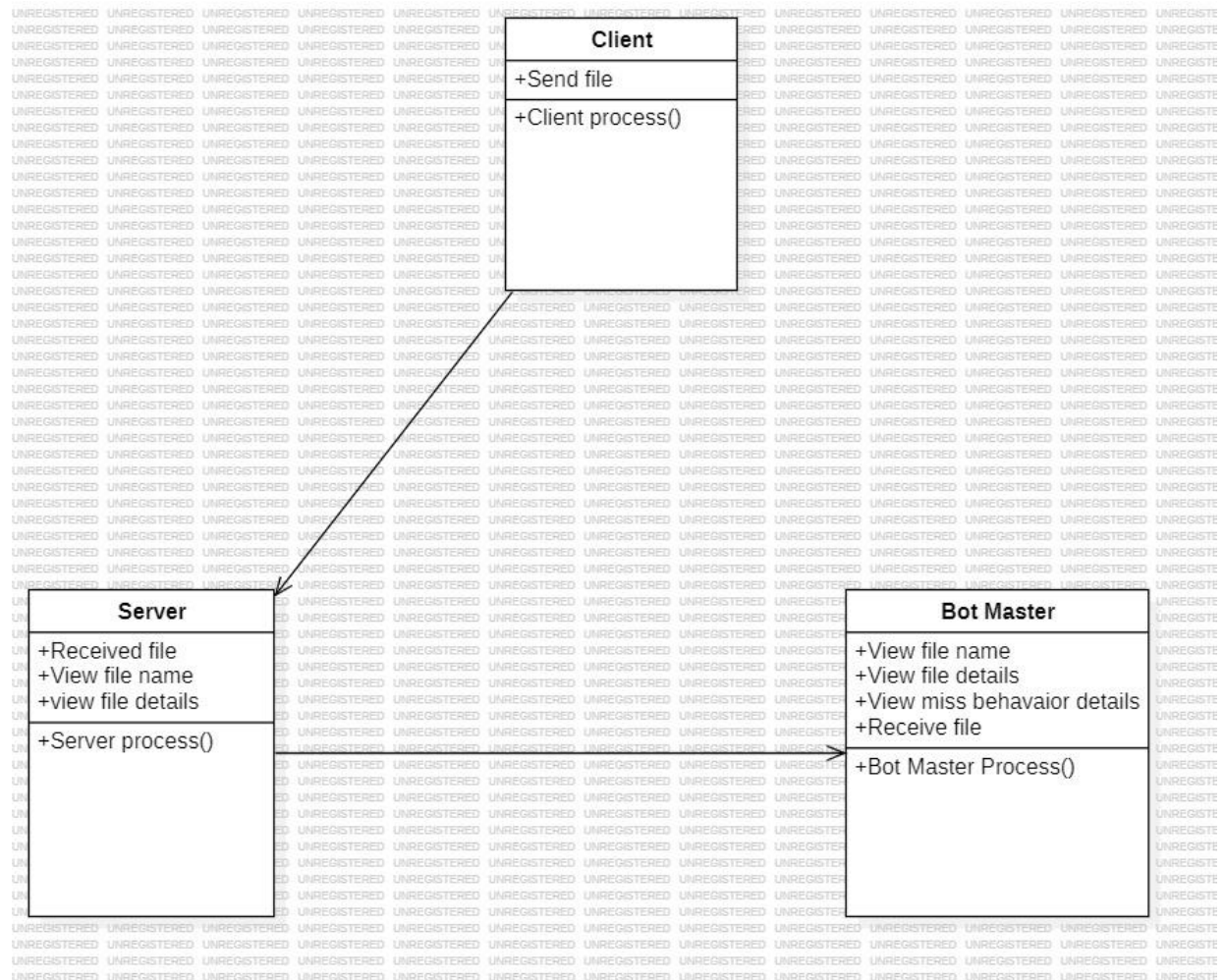


Fig 4.3. Class Diagram

4.4 OBJECT DIAGRAM:

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams. Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

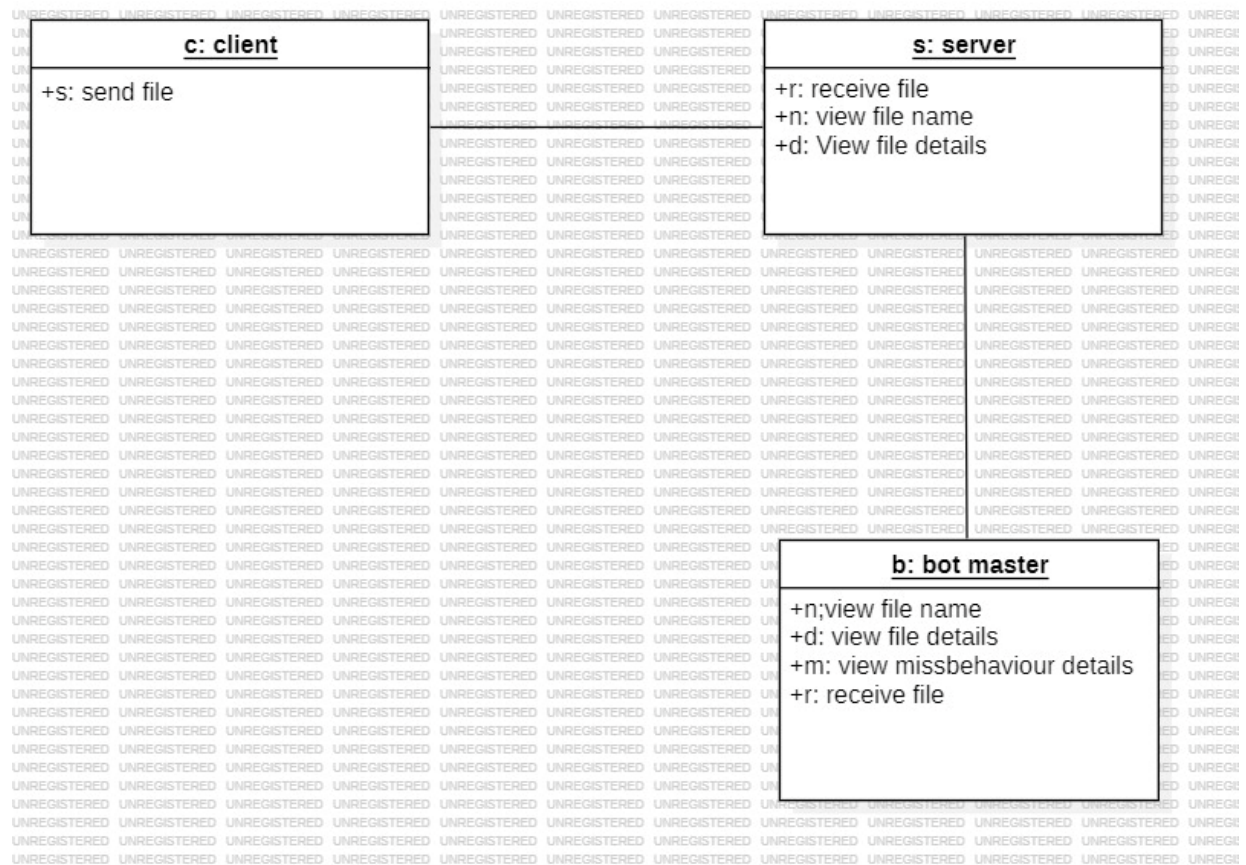


Fig 4.4. Object Diagram

4.5 COMPONENT DIAGRAM

UML component diagram are used in modelling the physical aspects of the object-oriented systems that are useful for visualizing, specifying and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system, components that are often used to model the static implementation view of a system

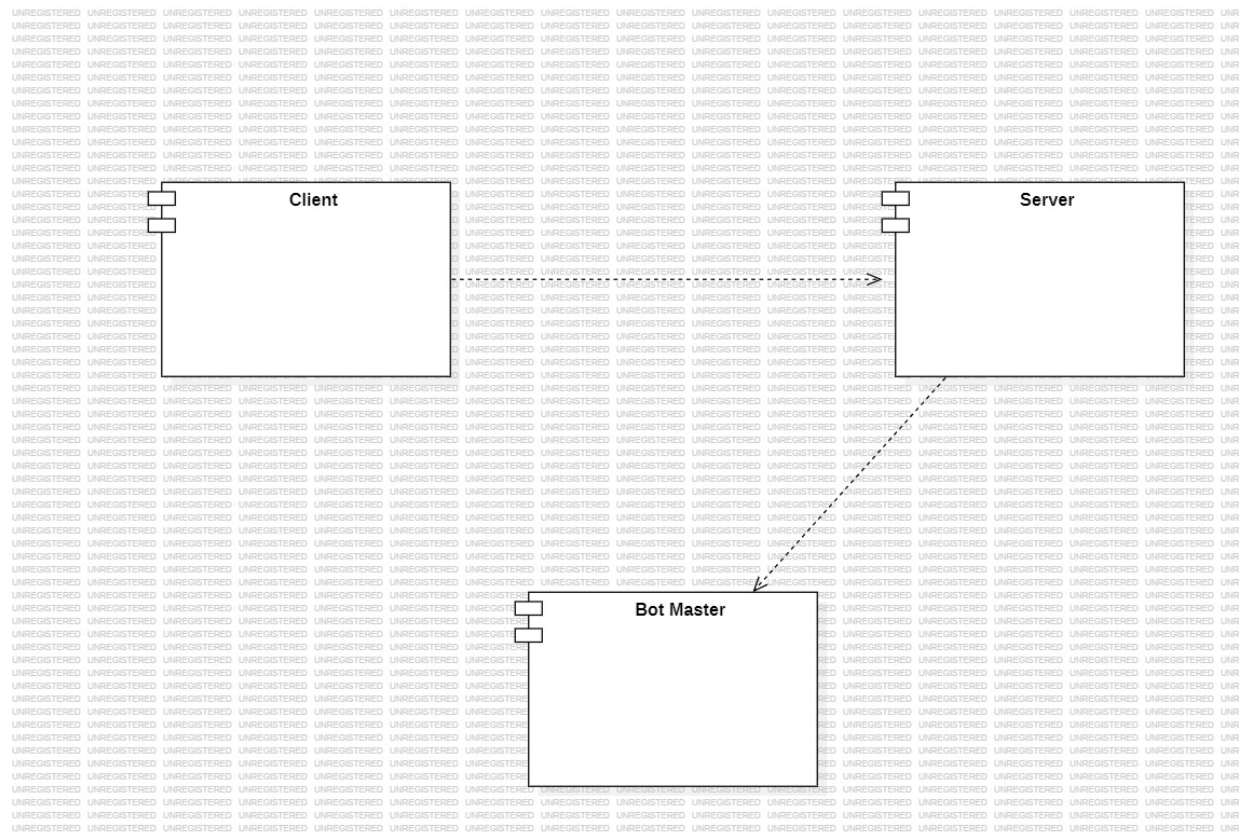


Fig 4.5. Component Diagram

DATA FLOW DIAGRAM

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.
5. Representations:
 1. Entity or source
 2. Data flow
 3. Process

```

graph TD
    Compile[Compile] --> Run[Run RMI]
    Run --> Client[Client]
    Client -. "Send file" .-> Server[Server]
    Server -. "Receive file" .-> login[login]
    login -. "View file name & details" .-> BotMaster[Bot Master]
    BotMaster -. "Receive File" .-> login
    BotMaster -. "View missbehaviour details" .-> login
    login -. "View file name & details" .-> BotMaster
  
```

Fig 4.6. Data Flow Diagram

CHAPTER-5

CODING

SOURCE CODE

Botmaster:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.*;

import java.net.*;

import java.util.*;

import javax.swing.event.*;

import java.sql.*;

import java.rmi.*;

import org.jvnet.substance.*;


public class botmaster extends JFrame implements ActionListener {


    String filesel, s1, s2, mst;

    String[] pmes;

    String[] str1 = new String[10];

    Random r;

    File f;


    JLabel fname;

    JLabel file;

    JLabel packet;
```

```
JLabel title, title1;
```

```
JButton view;
```

```
JButton send;
```

```
JButton mis;
```

```
JButton detail;
```

```
TextField fname1;
```

```
TextField file1;
```

```
TextArea packet1;
```

```
String msg = "";
```

```
Container c;
```

```
ImageIcon ii;
```

```
ImageIcon i2;
```

```
Socket c1;
```

```
static {
```

```
    try {
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentWatermark("org.jvnet.substance.watermark.SubstanceMagneticFieldWatermark");
```

SubstanceLookAndFeel

```
.setCurrentTheme("org.jvnet.substance.theme.SubstanceCharcoalTheme");
```

SubstanceLookAndFeel

```
.setCurrentGradientPainter("org.jvnet.substance.painter.SpecularGradientPainter");
```

SubstanceLookAndFeel

```
.setCurrentButtonShaper("org.jvnet.substance.button.BaseButtonShaper");
```

```
UIManager.setLookAndFeel(new SubstanceLookAndFeel());
```

```
    } catch (Exception e) {
```

```
        // TODO: handle exception
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

```
public botmaster()
```

```
{
```

```
    initializeComponent();
```

```
}
```



```

private void initializeComponent()

{

    // new net();

    // v = net.vnode;

    fname = new JLabel("<html><font color=#FFFFFF size=+1><strong>Search  
Query</strong></font>");

    file = new JLabel("<html><font color=#FFFFFF  
size=+1><strong>File</strong></font>");

    packet = new JLabel("<html><font color=#FFFFFF size=+1><strong>Packet  
Sending</strong></font>");

    title = new JLabel("<html><font color=#FFFFFF size=+2><strong>BOT NET  
ATTACK IN </strong></font>");

    title1 = new JLabel("<html><font color=#FFFFFF  
size=+2><strong>COMPUTER NETWORK SECURITY</strong></font>");


    JLabel imageLabel1 = new JLabel();

    ImageIcon v1 = new ImageIcon(this.getClass().getResource("archil.jpg"));

    imageLabel1.setIcon(v1);


    view = new JButton("View Files");

    view.addActionListener(this);

    send = new JButton("Receive");

    send.addActionListener(this);

    mis = new JButton("Misbehavior Details");

    mis.addActionListener(this);

```

```
detail = new JButton("File Details");
```

```
detail.addActionListener(this);
```

```
fname1 = new JTextField(10);
```

```
file1 = new JTextField(10);
```

```
packet1 = new JTextArea();
```

```
JScrollPane sa = new JScrollPane(packet1);
```

```
c = getContentPane();
```

```
c.setLayout(null);
```

```
c.add(fname);
```

```
// c.add(file);
```

```
// c.add(packet);
```

```
c.add(sa);
```

```
c.add(imageLabel1);
```

```
c.add(title);
```

```
c.add(title1);
```

```
c.add(view);
```

```
c.add(send);
```

```
c.add(mis);
```

```
c.add(detail);
```

```
c.add(fname1);

// c.add(file1);


c.setBackground(new java.awt.Color(100, 90, 200));

c.setLocation(300, 300);


title.setBounds(75, 30, 700, 50);

title1.setBounds(300, 80, 500, 50);


imageLabel1.setBounds(370, 250, 350, 300);


fname.setBounds(295, 175, 250, 25);

file.setBounds(30, 200, 250, 25);

packet.setBounds(170, 300, 200, 25);

sa.setBounds(62, 225, 150, 150);


view.setBounds(75, 175, 125, 25);

send.setBounds(465, 225, 125, 25);

mis.setBounds(60, 425, 155, 25);

detail.setBounds(60, 465, 155, 25);


fname1.setBounds(465, 175, 150, 25);

file1.setBounds(150, 200, 100, 25);
```

```

        setSize(800, 600);

        setVisible(true);

        setTitle("botmaster");

    }

    public void actionPerformed(ActionEvent ae)

    {

        if (ae.getSource() == view) {

            try {

                String url = "rmi://127.0.0.1/server";

                srvint in = (srvint) Naming.lookup(url);

                String v = "view";

                StringBuffer sb = (in.viewfile(v));

                packet1.setText(sb.toString());

            } catch (Exception e) {

                System.out.println(e);

```

```
    }  
}
```

```
if (ae.getSource() == detail) {
```

```
    new detail();  
}
```

```
if (ae.getSource() == send) {
```

```
    if (fname1.getText().equals("")) {
```

```
        JOptionPane.showMessageDialog(null, "Enter File Name");
```

```
        fname1.requestFocus();
```

```
    } else {
```

```
        String fn = fname1.getText();
```

```
        try {
```

```
            String url = "rmi://127.0.0.1/server";
```

```
            srvint in = (srvint) Naming.lookup(url);
```

```
            byte[] b1 = in.storefile(fn);
```

```
String file, dir, path, ms;

FileDialog fd2 = new FileDialog(this, "SAVE",
FileDialog.SAVE);

fd2.setVisible(true);

file = fd2.getFile();

dir = fd2.getDirectory();

path = dir + file;

try {

    FileOutputStream fos = new
FileOutputStream(path);

    for (int k = 0; k <= b1.length; k++) {

        fos.write(b1[k]);

    }

} catch (Exception e) {

}

}
```

Server:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.*;

import java.net.*;

import java.util.*;

import java.util.Date;

import java.text.SimpleDateFormat;

import javax.swing.event.*;

import java.sql.*;

import java.rmi.*;

import java.security.InvalidKeyException;

import java.security.Key;

import org.jvnet.substance.*;

public class server {

    public static void main(String[] args) throws Exception {

        srvimp rob = new srvimp();

        Naming.rebind("server", rob);

        System.out.println("Server is waiting for Client to Connect:");

        storagewindow sw = new storagewindow();

    }
```

```
}
```

```
class storagewindow extends JFrame implements ActionListener
```

```
{
```

```
    JLabel vfile;
```

```
    JLabel file;
```

```
    JLabel packet;
```

```
    JLabel fid;
```

```
    JLabel title, title1;
```

```
    JLabel n1;
```

```
    JButton open;
```

```
    JButton detail;
```

```
    JButton view;
```

```
    JTextField fname1;
```

```
    JTextField file1;
```

```
    JTextField fid1;
```

```
    JTextArea packet1;
```



```
JTextArea packet2;
```

```
String msg = "";
```

```
String filesel;
```

```
Container c;
```

```
ImageIcon ii;
```

```
ImageIcon i2;
```

```
static {
```

```
    try {
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentWatermark("org.jvnet.substance.watermark.SubstanceMagneticFieldWatermark");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentTheme("org.jvnet.substance.theme.SubstanceCharcoalTheme");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentGradientPainter("org.jvnet.substance.painter.SpecularGradientPainter");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentButtonShaper("org.jvnet.substance.button.BaseButtonShaper");
```

```
        UIManager.setLookAndFeel(new SubstanceLookAndFeel());
```

```

        // new net();

        // v = net.vnode;

        vfile = new JLabel("<html><font color=#FFFFFF size=+1><strong>All
Files</strong></font>");

        fid = new JLabel("<html><font color=#FFFFFF size=+1><strong>File
ID:</strong></font>");

        file = new JLabel("<html><font color=#FFFFFF
size=+1><strong>File</strong></font>");

        packet = new JLabel("<html><font color=#FFFFFF size=+1><strong>Packet
Sending</strong></font>");

        title = new JLabel("<html><font color=#FFFFFF size=+2><strong>BOT NET
ATTACK IN</strong></font>");

        title1 = new JLabel("<html><font color=#FFFFFF
size=+2><strong>COMPUTER NETWORK SECURITY </strong></font>");

        n1 = new JLabel("<html><font color=#FFFFFF size=+1><strong>Storage
Node</strong></font>");


        JLabel imageLabel1 = new JLabel();

        ImageIcon v1 = new
ImageIcon(this.getClass().getResource("AnimatedTower.gif"));

        imageLabel1.setIcon(v1);


        open = new JButton("All Files");

        open.addActionListener(this);

        detail = new JButton("Details");

        detail.addActionListener(this);

        view = new JButton("View");

```

```
view.addActionListener(this);
```

```
fname1 = new JTextField(10);
```

```
file1 = new JTextField(10);
```

```
fid1 = new JTextField(10);
```

```
packet1 = new JTextArea();
```

```
packet2 = new JTextArea(6, 20);
```

```
JScrollPane sa = new JScrollPane(packet2);
```

```
JScrollPane sa1 = new JScrollPane(packet1);
```

```
c = getContentPane();
```

```
c.setLayout(null);
```

```
// c.add(vfile);
```

```
// c.add(file);
```

```
// c.add(packet);
```

```
c.add(sa1);
```

```
c.add(sa);
```

```
c.add(n1);
```

```
c.add(fid);
```

```
c.add(fid1);
```

```
c.add(imageLabel1);
```

```
c.add(title);
```

```
c.add(title1);
```

```
c.add(open);
```

```
c.add(detail);
```

```
c.add(view);
```

```
// c.add(fname1);
```

```
// c.add(file1);
```

```
c.setBackground(new java.awt.Color(100, 90, 200));
```

```
c.setLocation(300, 300);
```

```
title.setBounds(75, 30, 700, 50);
```

```
title1.setBounds(300, 80, 500, 50);
```

```
fid.setBounds(250, 430, 200, 50);
```

```
fid1.setBounds(345, 445, 100, 25);
```

```
imageLabel1.setBounds(305, 185, 350, 150);
```

```
vfile.setBounds(100, 150, 250, 25);
```

```
file.setBounds(30, 200, 250, 25);
```

```
packet.setBounds(150, 300, 200, 25);
```

```
sa1.setBounds(65, 185, 150, 150);
```

```
sa.setBounds(500, 185, 200, 150);
```

```
n1.setBounds(305, 315, 150, 100);
```

```
open.setBounds(77, 145, 125, 25);
```

```
detail.setBounds(533, 145, 125, 25);
```

```
view.setBounds(345, 495, 75, 25);
```

```
private JLabel label1;
```

```
private JTextField text1;
```

```
private JButton submit;
```

```
Container c;
```

```
String a1;
```

```
admin1(String a) {
```

```
    this.a1 = a;
```

```
    c = getContentPane();
```

```
    c.setLayout(new FlowLayout());
```

```
    label1 = new JLabel("Enter Password");
```

```
c.add(label1);
```

```
text1 = new JTextField(10);
```

```
text1.addActionListener(this);
```

```
c.add(text1);
```

```
submit = new JButton("submit");
```

```
submit.addActionListener(this);
```

```
c.add(submit);
```

```
add(submit);
```

```
submit.setBounds(160, 100, 100, 20);
```

```
setLayout(null);
```

```
add(label1);
```

```
label1.setBounds(40, 40, 120, 25);
```

```
add(text1);
```

```
text1.setBounds(160, 40, 140, 20);
```

```
this.setSize(400, 175);
```

```
this.show();
```

```
}
```

```
public void actionPerformed(ActionEvent ae)
```

```

{

    this.dispose();

    String pa = "jp";

    if (ae.getSource() == submit) {

        if (text1.getText().equals("")) {

            JOptionPane.showMessageDialog(null, "Enter Password");

            text1.requestFocus();

        } else if (text1.getText().equals(pa)) {

            // if password correct show original file

            String fnam1 = "asdas";

            byte[] b1 = null;

            try {

                Class.forName("com.mysql.jdbc.Driver");

                Connection con =

DriverManager.getConnection("jdbc:mysql://localhost:3306/dbase", "root", "");

                Statement st = con.createStatement();

                String sql = "select * from file where filename=" + a1 +

"";

```

```

        ResultSet rs = st.executeQuery(sql);

        while (rs.next()) {

            b1 = rs.getBytes("file");

        }

    } catch (Exception e) {

Container c;

byte[] ba;

String na;

viewfile(String n, byte[] a) {

    this.na = n;

    this.ba = a;

    c = getContentPane();

    c.setLayout(new FlowLayout());

    box = new JTextArea(25, 40);

    sa = new JScrollPane(box);

    sa.setBounds(20, 20, 500, 500);

    c.add(sa);

    String ba1 = new String(ba);

    String x = encryptString(ba1);

```



```
        box.append(x);

        this.setTitle(na + ".txt");

        this.setSize(500, 500);

        this.show();

    }
```

```
public static String encryptString(String str) {

    StringBuffer sb = new StringBuffer(str);

    String key = "Encrypt Key";

    int lenStr = str.length();

    int lenKey = key.length();

    //

    // For each character in our string, encrypt it...

    for (int i = 0, j = 0; i < lenStr; i++, j++) {

        if (j >= lenKey)

            j = 0; // Wrap 'round to beginning of key string.

        //

        // XOR the chars together. Must cast back to char to avoid compile error.

        //

        sb.setCharAt(i, (char) (str.charAt(i) ^ key.charAt(j)));
    }
}
```

```
    }

    return sb.toString();
}

public void itemStateChanged(ItemEvent ie) {
    repaint();
}

public static void main(String arg[]) {

}
}
```

Client 1:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.event.*;
import java.sql.*;
import java.rmi.*;
```

```
import org.jvnet.substance.*;
```

```
public class client1 extends JFrame implements ActionListener
```

```
{
```

```
    String filesel, s1, s2, mst;
```

```
    String[] pmes;
```

```
    String[] str1 = new String[10];
```

```
    Random r;
```

```
    File f;
```

```
    JLabel fname;
```

```
    JLabel file;
```

```
    JLabel packet;
```

```
    JLabel title, title1;
```

```
    JLabel n1;
```

```
    JButton open;
```

```
    JButton send;
```

```
    JButton reset;
```

```
    JTextField fname1;
```

```
    JTextField file1;
```

```
JTextArea packet1;
```

```
String msg = "";
```

```
Container c;
```

```
ImageIcon ii;
```

```
ImageIcon i2;
```

```
Socket c1;
```

```
static {
```

```
    try {
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentWatermark("org.jvnet.substance.watermark.SubstanceMagneticFieldWatermark");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentTheme("org.jvnet.substance.theme.SubstanceCharcoalTheme");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentGradientPainter("org.jvnet.substance.painter.SpecularGradientPainter");
```

```
        SubstanceLookAndFeel
```

```
        .setCurrentButtonShaper("org.jvnet.substance.button.BaseButtonShaper");
```

```
        UIManager.setLookAndFeel(new SubstanceLookAndFeel());
```

```

        } catch (Exception e) {

            // TODO: handle exception

            e.printStackTrace();

        }

    }

    public client1()

    {

        initializeComponent();

    }

    private void initializeComponent()

    {

        fname = new JLabel("<html><font color=#FFFFFF  

size=+1><strong>FileName</strong></font>");

        file = new JLabel("<html><font color=#FFFFFF  

size=+1><strong>File</strong></font>");

        packet = new JLabel("<html><font color=#FFFFFF size=+1><strong>Packet  

Sending</strong></font>");

```

```
title = new JLabel("<html><font color=#FFFFFF size=+2><strong>BOT NET  
ATTACK IN COMPUTER </strong></font>");
```

```
title1 = new JLabel("<html><font color=#FFFFFF  
size=+2><strong>NETWORK SECURITY </strong></font>");
```

```
n1 = new JLabel("<html><font color=#FFFFFF  
size=+1><strong>client1 </strong></font>");
```

```
JLabel imageLabel1 = new JLabel();
```

```
ImageIcon v1 = new  
ImageIcon(this.getClass().getResource("AnimatedTower.gif"));
```

```
imageLabel1.setIcon(v1);
```

```
open = new JButton("Open");
```

```
open.addActionListener(this);
```

```
send = new JButton("Send");
```

```
send.addActionListener(this);
```

```
fname1 = new JTextField(10);
```

```
file1 = new JTextField(10);
```

```
packet1 = new JTextArea();
```

```
c = getContentPane();
```

```
c.setLayout(null);
```

```
c.add(fname);
```

```
c.add(file);
```

```
c.add(packet);
```

```
c.add(packet1);
```

```
c.add(n1);
```

```
c.add(imageLabel1);
```

```
c.add(title);
```

```
c.add(title1);
```

```
c.add(open);
```

```
c.add(send);
```

```
c.add(fname1);
```

```
c.add(file1);
```

```
c.setBackground(new java.awt.Color(100, 90, 200));
```

```
c.setLocation(300, 300);
```

```
title.setBounds(75, 30, 700, 50);
```

```
title1.setBounds(300, 80, 500, 50);
```

```
imageLabel1.setBounds(475, 125, 450, 350);
```

```
fname.setBounds(30, 150, 250, 25);
```

```
file.setBounds(30, 200, 250, 25);  
packet.setBounds(150, 300, 200, 25);  
packet1.setBounds(150, 350, 150, 150);  
n1.setBounds(475, 375, 150, 100);
```

```
open.setBounds(255, 200, 75, 25);  
send.setBounds(150, 245, 75, 25);
```

```
fname1.setBounds(150, 150, 150, 25);  
file1.setBounds(150, 200, 100, 25);
```

```
setSize(800, 600);  
setVisible(true);  
setTitle("Client 1");
```

```
}
```

```
public void actionPerformed(ActionEvent ae)
```

```
{
```

```
    if (ae.getSource() == send) {
```

```
        if (fname1.getText().equals("")) {
```



```
JOptionPane.showMessageDialog(null, "Enter File Name");

fname1.requestFocus();

} else if (file1.getText().equals("")) {

    JOptionPane.showMessageDialog(null, "Browse File");

    file1.requestFocus();

} else {

    System.out.println("Connected To server....");

    System.out.println("\n");

    String nm = "client1";

    String fn = fname1.getText();

    f = new File(filesel);

    String st;

    try {

        String url = "rmi://127.0.0.1/server";

        srvint in = (srvint) Naming.lookup(url);
```

```
int sz;
```

```
byte[] buffer;
```

```
pmes = new String[10];
```

```
r = new Random();
```

```
FileInputStream fin = new FileInputStream(filesel);
```

```
sz = fin.available() / 7;
```

```
buffer = new byte[sz];
```

```
int m = r.nextInt(7);
```

```
StringBuffer sb = new StringBuffer();
```

```
for (int i = 0; i <= 7; i++) {
```

```
    fin.read(buffer);
```

```
    st = new String(buffer);
```

```
    pmes[i] = st;
```

```
    sb.append(st);
```

```
}
```

```
String pt = packet1.getText();
```

```
String pt1 = "null";
```

```

        if (pt != pt1) {

            packet1.setText("");

        }

        System.out.println("File Sending From client1 to
server");

        System.out.println("\n");

        System.out.println("Filename:\t" + fn);

        System.out.println("\n");

        for (int i = 0; i <= 7; i++) {

            if (i != m) {

                packet1.append("Packet[" + i +

                "]:PACK");

                packet1.append("\n");

                new ccl(i, m);

            } else {

                packet1.append("Packet[" + i +

                "]:NACK");

                packet1.append("\n");

```

```

                                new cc1(i, m);
                                }

                                }

                                System.out.println("\n \t Packet[" + m + "]:Positive
ACK");

                                System.out.println("\n File Sent To server");
                                packet1.append("Packet[" + m + "]:PACK");

                                in.getfile(nm, fn, sb);
                                file1.setText("");
                                fname1.setText("");

                                } catch (Exception e) {
                                }

                                }

                                }

                                if (ae.getSource() == open)

                                {

```

```

        FileDialog fd1 = new FileDialog(this, "OPEN", FileDialog.LOAD);

        fd1.setVisible(true);

        filesel = fd1.getDirectory() + fd1.getFile();

        file1.setText(filesel);

    }

}

public static void main(String args[]) {

    new client1();

    System.out.println("client1 Started....");

}

}

class cc1 extends Thread {

    Thread t;

    cc1(int ps, int rm) {

        try {

            if (rm != ps) {

                System.out.println("\t Packet[" + ps + "]:Positive ACK");

```

```

        System.out.println("\n");

        t.sleep(2000);

    } else {

        System.out.println("Packet[" + ps + "]:Negative ACK");

        System.out.println("\n");

    }

} catch (Exception e) {

    System.out.println(e);

} } }

```

Login:

```

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.*;

import java.net.*;

import java.util.*;

import javax.swing.event.*;

import java.sql.*;

import java.rmi.*;

import org.jvnet.substance.*;

```

```
public class login extends JFrame implements ActionListener
```

```
{
```

```
    JLabel fname;
```

```
    JLabel file;
```

```
    JLabel packet;
```

```
    JLabel title, title1;
```

```
    JLabel n1, n2;
```

```
    JButton open;
```

```
    JButton send;
```

```
    JButton reset;
```

```
    JTextField fname1;
```

```
    JPasswordField file1;
```

```
    JTextArea packet1;
```

```
    String msg = "";
```

```
    Container c;
```

```
    ImageIcon ii;
```

```
ImageIcon i2;
```

```
Socket c1;
```

```
static {
```

```
    try {
```

```
        SubstanceLookAndFeel
```

```
    }
```

```
private void initComponents()
```

```
{
```

```
        fname = new JLabel("<html><font color=#FFFFFF size=+1><strong>User  
Name</strong></font>");
```

```
        file = new JLabel("<html><font color=#FFFFFF  
size=+1><strong>Password</strong></font>");
```

```
        title = new JLabel("<html><font color=#FFFFFF size=+2><strong>ADMIN  
LOGIN </strong></font>");
```

```
        JLabel imageLabel1 = new JLabel();
```

```
        ImageIcon v1 = new ImageIcon(this.getClass().getResource(""));
```

```
        imageLabel1.setIcon(v1);
```



```
send = new JButton("Send");  
  
send.addActionListener(this);  
  
fname1 = new JTextField(10);  
  
file1 = new JPasswordField(10);
```

```
c = getContentPane();  
  
c.setLayout(null);  
  
fname1.setBounds(150, 100, 150, 25);  
  
file1.setBounds(150, 150, 150, 25);
```

```
setSize(350, 300);  
  
setVisible(true);  
  
setTitle("BOT Admin Login");
```

```
}
```

```
public void actionPerformed(ActionEvent ae)
```

```
{
```

```
    if (ae.getSource() == send) {
```

```
        if (fname1.getText().equals("")) {
```

```

        JOptionPane.showMessageDialog(null, "Enter UserName");

        fname1.requestFocus();

    } else if (file1.getText().equals("")) {

        JOptionPane.showMessageDialog(null, "Password");

        file1.requestFocus();

    } else if ((fname1.getText().equals("bot")) &&
(file1.getText().equals("bot"))) {

        new botmaster();

        this.dispose();

    }

    else {

        JOptionPane.showMessageDialog(null, "Invalid UserName and
Password");

        fname1.requestFocus();

    }

}

```

```
}
```

```
public static void main(String args[]) {
```

```
    new login();
```

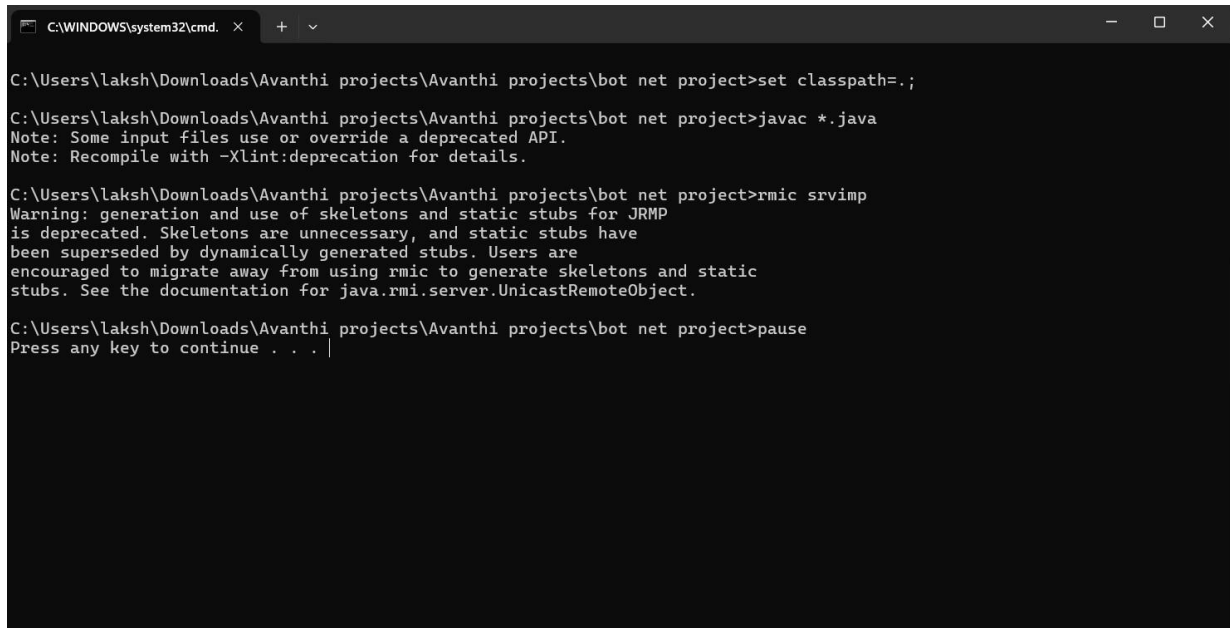
```
}
```

```
}
```

CHAPTER-6

OUTPUTS

Step 1: First we will run the compile and press A key to continue



```
C:\WINDOWS\system32\cmd. x + v
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>set classpath=.;
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>javac *.java
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>rmic srvimp
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>pause
Press any key to continue . . . |
```

Fig 6.1 : compile in command prompt

Step 2: Then we run RMI (Remote Method Inogation) to connect the server

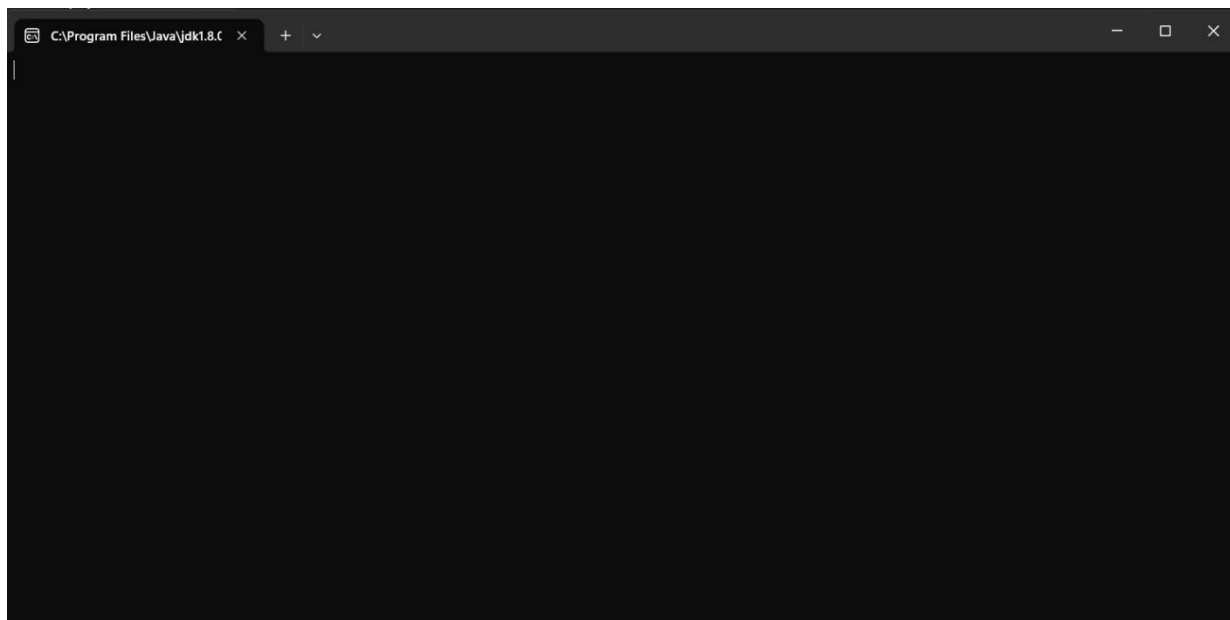


Fig 6.2 : Without any Error RMI like this

Step 3: After that we open client-1 in cmd

```
C:\WINDOWS\system32\cmd. x + v
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>set classpath=mysql-connector-java-3.1.14-bin.jar;
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>java client1
client1 Started....
Connected To server....

File Sending From client1 to server

Filename:      karthik

    Packet[0]:Positive ACK

    Packet[1]:Positive ACK

    Packet[2]:Positive ACK

    Packet[3]:Positive ACK

Packet[4]:Negative ACK

    Packet[5]:Positive ACK

    Packet[6]:Positive ACK

    Packet[7]:Positive ACK

    Packet[4]:Positive ACK

File Sent To server
```

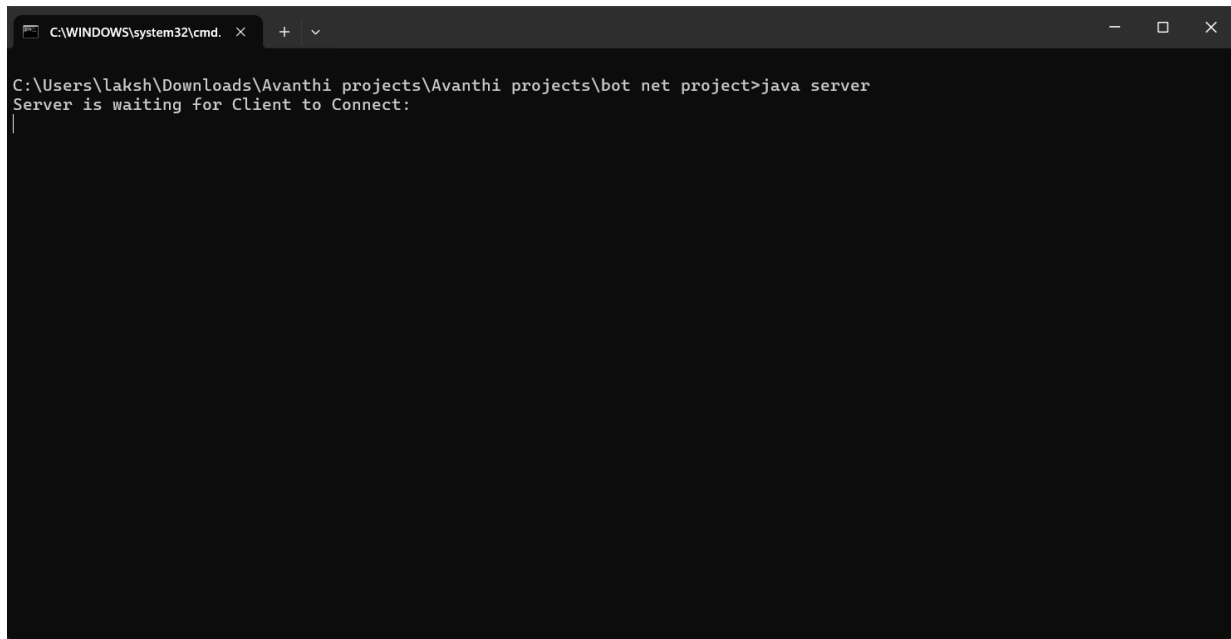
Fig 6.3

Step4: Then open client 1 interface open like this



Fig 6.4

Step 6: After that We start a server to client connection



```
C:\WINDOWS\system32\cmd. x + v
C:\Users\laksh\Downloads\Avanathi projects\Avanathi projects\bot net project>java server
Server is waiting for Client to Connect:
```

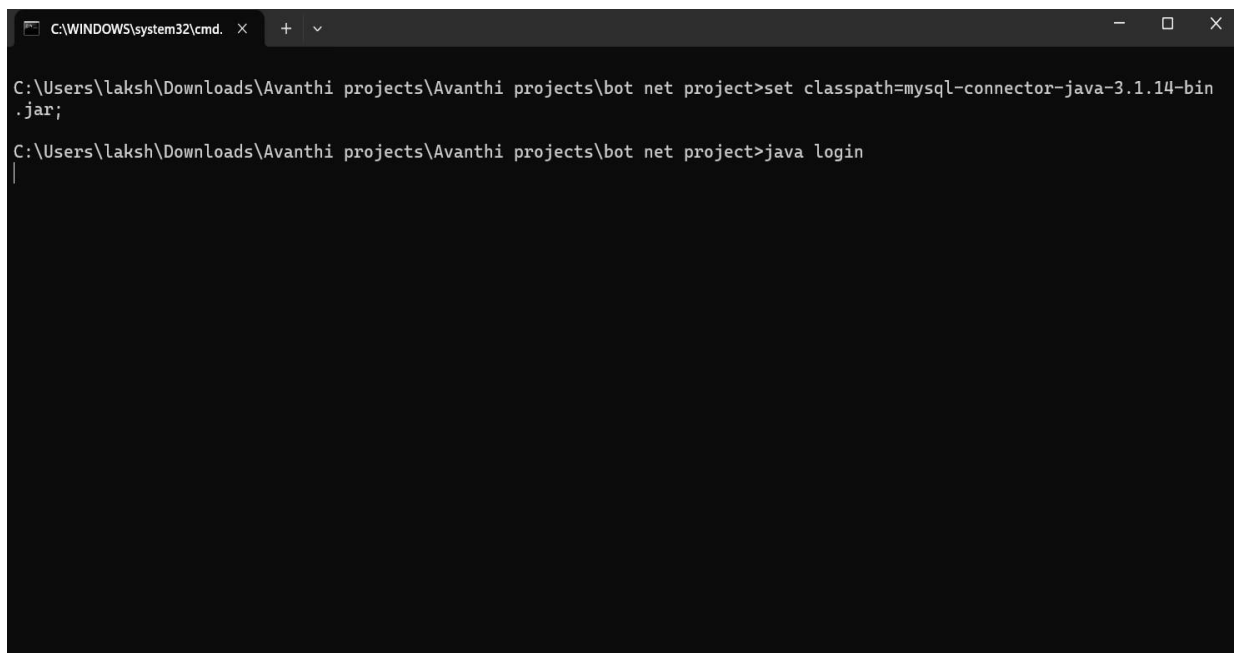
Fig 6.5

Step 7:



Fig 6.6

Step 8:



```
C:\WINDOWS\system32\cmd. X + v
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>set classpath=mysql-connector-java-3.1.14-bin.jar;
C:\Users\laksh\Downloads\Avanthi projects\Avanthi projects\bot net project>java login
```

Fig 6.7

Step 9:

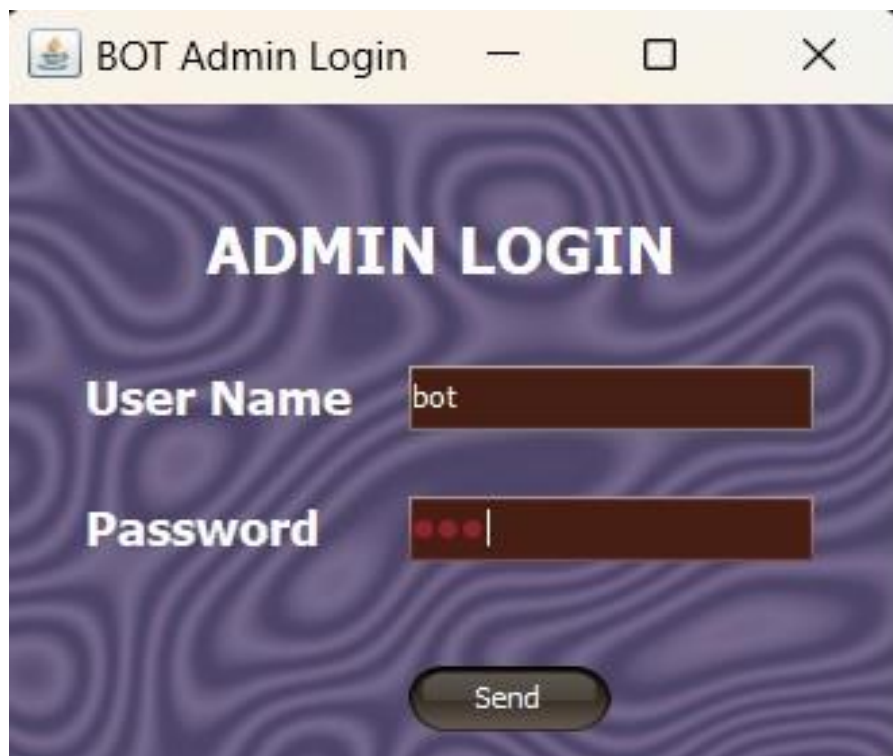


Fig 6.8

Step 10:

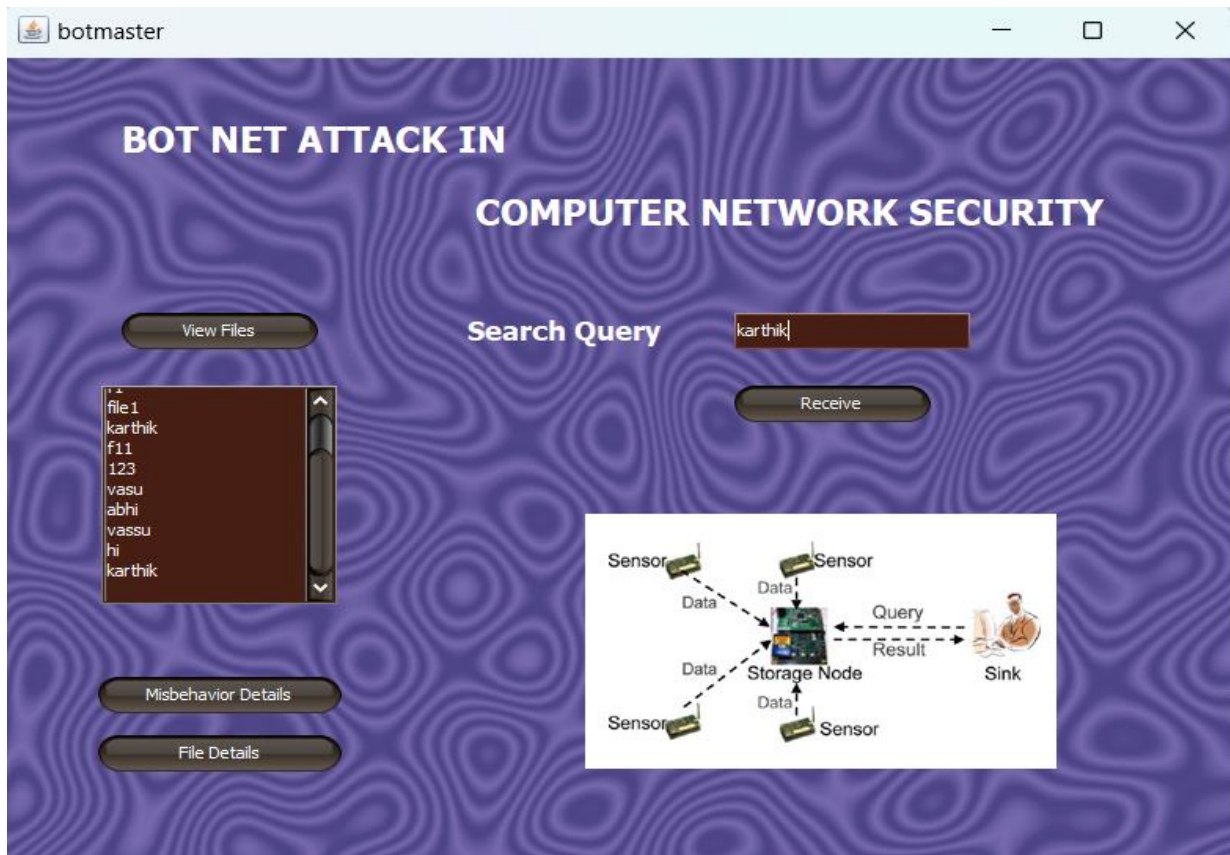


Fig 6.9

CHAPTER-7

TESTING

SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product it is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

CHAPTER-8
CONCLUSION

CONCLUSION

As well discussed above since 1988, Botnet have evolved from the beginning assistant tool to the predominant threat in modern internet and as discussed in this paper, in 1988 Botnet was not a malicious activity but later in 1998 , attacker use the bot to perform malicious activity via cyber crime. That is Botnets pose a significant and growing threat against cyber-security **as they** provide a key platform for many cybercrimes such as DDoS attacks against critical targets, malware dissemination, phishing, and click fraud etc. Although the number of bots to each Botnet seems to be decreasing, the monetary damaging power of the Botnets is continuously increasing given the development of internet bandwidth due to change in technology.

This study is focused on the attacks that a botmaster attempts to steal sensitive data from the victim machines and we can spread our tracing pebbles along with the stolen data all the way back to the botmaster. However, if a botmaster only wants to communicate with the victims, more intelligence has to be integrated in the Pebbleware. As most of the botnet traffic is encrypted by symmetric keys for efficiency, we only study symmetric key identification.

Asymmetric key identification is a challenging research topic in general. However, our results on symmetric key identification and the specific botnet application environment may shed light on future investigation.

Anonymous network and social network services, such as Twitter and Facebook, might be moved into clouds. Their security, particular defense against botnets, is an intriguing research topic.

CHAPTER-9
FUTURE SCOPE

FUTURE SCOPE

Based on the conclusion section of the paper, some potential future research directions and scope mentioned include:

1. Investigating asymmetric key identification techniques for botnet communications, as the paper focused mainly on symmetric key identification.
2. Studying defense mechanisms against botnets that leverage anonymous networks and social networking services like Twitter and Facebook as botnet communication channels, especially as these services move to cloud environments.
3. Further enhancing the Pebbleware tracing technique to better handle scenarios where the botmaster only wants to communicate with victims rather than stealing data, by integrating more intelligence into the Pebbleware.
4. Exploring new zero-day vulnerabilities in botnet command and control servers and client software that could enable implementing the Pebbleware and tracing back to the botmaster across stepping stones.
5. Developing more advanced key identification methods to handle botnets using non-standard or sophisticated encryption schemes that don't follow typical patterns.
6. Keeping pace with evolving botnet architectures, communication protocols and attack vectors by extending the botnet detection, prevention and mitigation approaches proposed.

In summary, the key future scope areas mentioned are enhancing the key identification, tracing techniques like Pebbleware, uncovering new vulnerabilities, and continuously adapting the methods to counter the rapidly evolving botnet threats leveraging new technologies and techniques like anonymous networks, cloud services and advanced encryption.

CHAPTER-10
REFERENCES

REFERENCES

1. Hossein Rouhani Zeidanloo, Azizah Bt Manaf, Payam Vahdani, Farzaneh Tabatabaei, Mazdak Zamani, "Botnet Detection Based on Traffic Monitoring" IEEE transaction, 2010.
2. SANS Institute InfoSec Reading Room provided a description on "Bot & Botnet: An overview" research on topics in information security, 2003.
3. Generation of a robust Botnet capable of maintaining control of its remaining bots even after a substantial portion of the Botnet population has been removed by defenders.
4. S. Nagendra Prabhu, Kemal Sultan Abdo & Gashaw Bekele Kabtimer, „Introducing Proxy Cloud Storage Using Internet Information Services in University and Utilization Of Its Resources in the Academic Institution“ in Journal of Network communications and Emerging Technologies, Volume 8, issue 2, February 2018.
5. Hossein Rouhani Zeidanloo, Farhoud Hosseinpour, Farhood Farid Etemad, "New Approach for Detection of IRC and P2P Botnets" , International Journal of Computer and Electrical Engineering, Vol.2, No.6, December, 2010, 1793-8163.
6. Prabhu S, Chandrasekar V & Shanthi S, Improving the performance of IDS using Arbitrary Decision Tree in Network Security, International Journal of Advanced Science and Technology Vol. 29, No. 3, (2020), pp. 3453 - 3462
7. Hossein Rouhani Zeidanloo, Azizah Abdul Manaf, "Botnet Command and Control Mechanisms", IEEE transactions, 2009.
8. Nagendra Prabhu, S & Shanthi, D „A Survey on Anomaly Detection of Botnet in Network“ published in Volume 2, Issue 1 of International Journal of Advance Research in Computer Science and Management Studies in the year of 2014.
9. Robert F. Erbacher, Adele Cutler, Pranab Banerjee, Jim Marshall, "A Multi-Layered Approach to Botnet Detection", IEEE conference, 2010.
10. Nagendra Prabhu, S & Shanthi Saravanan, D, 2017, „An Efficient Botnet Detection System in Large Scenario Networks Using Adaptive Neuro Fuzzy Inference System Classifier“, Journal of Computational and Theoretical Nanoscience Vol. 14, 1–5, 2017