AthosCat

A lightweight and extensible network tool inspired by Netcat

When Netcat isn't enough, AthosCat tries... bravely.

Author: **Arthur Morain**

July 2025

Contents

1 Introduction

1.1 Presentation

AthosCat is a simplified reimplementation of Netcat, focusing on core features such as TCP and UDP client connections, server mode (listen), file transfer capabilities, and additional network utilities.

1.2 Motivations

Building upon my previous project, SecureSocketDH, which was my first experience in network programming, my goal is to deepen my understanding of network programming and to gain insight into the inner workings of the tools I often use.

This project also sits at the intersection of networking and cybersecurity. As many cybersecurity professionals know, Netcat is an indispensable network tool used for debugging, penetration testing, and secure communications. By reimplementing parts of Netcat, I aim to grasp both the practical and theoretical aspects of network interactions and security implications.

Moreover, developing AthosCat helps me practice critical programming skills such as socket management, protocol handling, and asynchronous I/O, which are fundamental for building robust and efficient networked applications.

For more details, the project repository is available at: AthosCat-Repo.

2 Client Mode Presentation

The client mode of AthosCat is designed to establish outbound connections to remote servers using either TCP or UDP protocols. It supports both IPv4 and IPv6 addressing, allowing flexible communication across diverse network environments.

2.1 Functionality

The client initiates a connection based on user-specified parameters such as target hostname or IP address, port number, and protocol type (TCP or UDP). For TCP, the client performs a standard socket connection to the server. For UDP, being a connectionless protocol, the client sends datagrams directly without establishing a persistent connection.

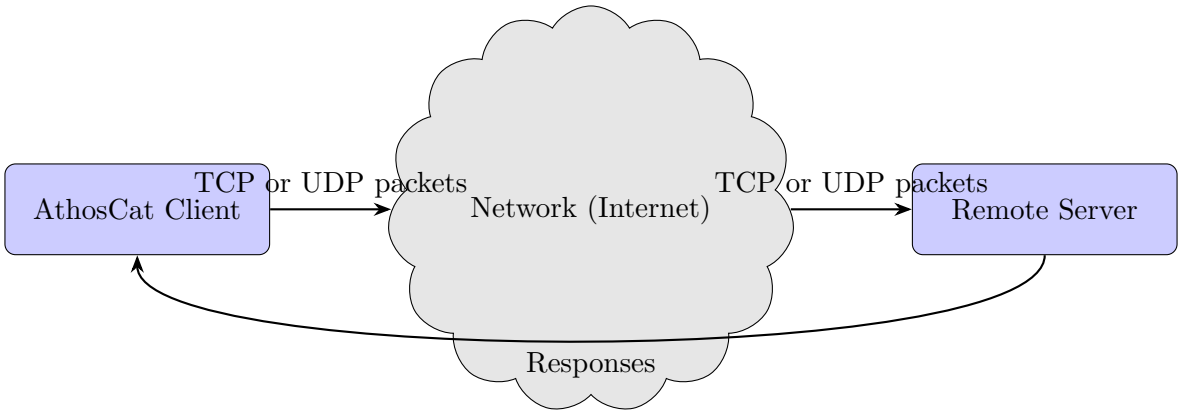
2.2 Implementation Details

The implementation leverages the POSIX socket API and supports I/O multiplexing through `select()`, allowing simultaneous monitoring of standard input and the socket descriptor. This enables seamless bidirectional communication where the user can send data via standard input and receive responses from the server concurrently.

Key features include:

- Protocol flexibility: TCP connection-oriented and UDP connectionless communication.
- IPv4 and IPv6 compatibility through address resolution with `getaddrinfo()`.
- Timeout handling via `select()`, ensuring the client does not block indefinitely.
- Graceful shutdown by detecting end-of-input and closing connections appropriately.

2.3 Representation



AthosCat Client	: Initiates connection, sends and receives data.
Network	: Represents the internet or local network.
Remote Server	: Listens on specified port and communicates.
TCP / UDP	: Transport protocols used for data exchange.

Figure 1: Detailed overview of AthosCat client connection to a remote server using TCP or UDP protocols.

2.4 Limitations and Possible Improvements

Currently, the client mode supports basic send and receive operations for TCP and UDP protocols. Possible future enhancements include:

- More robust error handling and recovery mechanisms.

- Implementation of additional features such as connection listening (server mode).
- Support for advanced networking features like encryption or proxying.
- Improved input/output buffering and user interface enhancements.

3 Listener Mode Presentation

The listener mode of AthosCat is designed to accept inbound connections from remote clients over TCP or to receive datagrams over UDP. It supports both IPv4 and IPv6, enabling it to operate flexibly in various network environments.

3.1 Functionality

In listener mode, AthosCat binds to a specified port and waits for incoming communication. For TCP, it listens for connection requests and accepts one client at a time, enabling bidirectional communication once connected. For UDP, it receives datagrams on the bound port without establishing a connection, reflecting the connectionless nature of the protocol.

3.2 Implementation Details

The listener utilizes the POSIX socket API for socket creation, binding, and communication. For TCP, it performs a `listen()` call with a backlog queue and uses `accept()` to handle incoming connections sequentially. For UDP, it uses `recvfrom()` in a loop to receive incoming datagrams. The implementation supports both IPv4 and IPv6 by resolving local addresses with `getaddrinfo()`. Timeout handling is optionally implemented via `setsockopt()` to avoid indefinite blocking on UDP sockets.

Key features include:

- TCP server functionality with connection acceptance and data exchange.
- UDP datagram reception on a specified port.
- IPv4 and IPv6 support via flexible address resolution.
- Optional socket receive timeout for UDP to prevent indefinite blocking.
- Verbose logging for connection and data reception events.

3.3 Representation

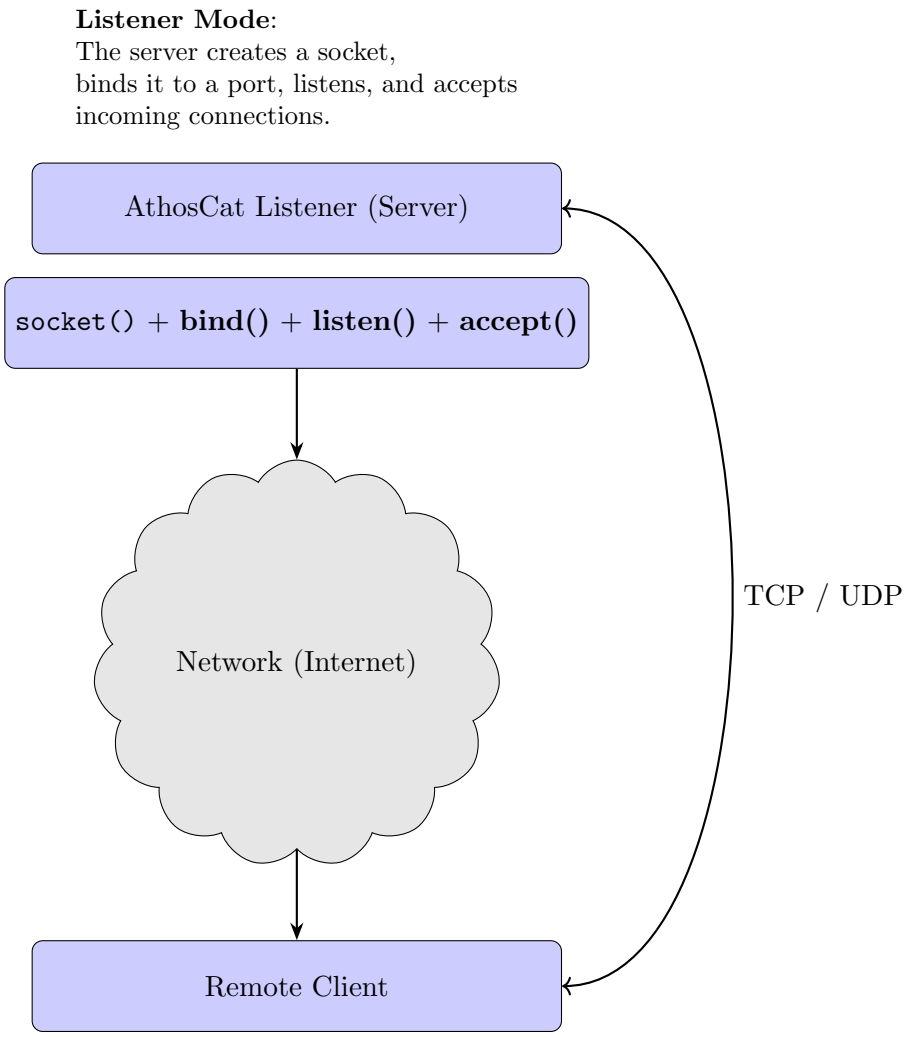


Figure 2: Schematic representation of AthosCat Listener mode.

3.4 Limitations and Possible Improvements

Currently, the listener mode supports handling a single TCP connection at a time and receiving UDP datagrams on a specified port. Possible future enhancements include:

- Support for handling multiple simultaneous TCP connections (multi-threading or asynchronous I/O).
- Improved error detection and recovery.
- Integration of security features such as TLS encryption.
- Enhanced logging and monitoring capabilities.

4 Using Redirections for File Transfer

One of AthosCat's strengths is its compatibility with standard input and output streams. This enables users to easily send and receive files using shell redirections, similar to how one would use traditional tools like `netcat`.

4.1 Sending a File (Client Side)

To send a file from the client to a remote listener, you can use input redirection with the `<` operator:

```
./athoscat --client -h <server_ip> -p <port> < file_to_send.txt
```

This command reads the contents of `file_to_send.txt` and sends it over the connection.

4.2 Receiving a File (Listener Side)

On the listener side, you can redirect the received data into a file using the `>` operator:

```
./athoscat --listener -p <port> > received_file.txt
```

This saves the incoming data into `received_file.txt`.

4.3 Example

- **Listener (Receiver)**:

  ```
  ./athoscat --listener -p 5000 > output.txt
  ```

- **Client (Sender)**:

  ```
  ./athoscat --client -h 192.168.1.42 -p 5000 < input.txt
  ```

In this example, the contents of `input.txt` will be transmitted from the client to the listener, and stored as `output.txt`.

4.4 Note on Binary Files

This technique works not only for text files but also for binary data. However, care should be taken to avoid character encoding issues when dealing with non-textual content. It is advised to run in environments that preserve byte fidelity (e.g., avoid text mode conversions on Windows).

5 Using the -e Option (Execute Command)

The `-e` option allows AthosCat to execute a local command after establishing a connection. This mimics the classic `netcat -e` functionality and can be used, for example, to launch a shell over a network connection.

5.1 How It Works

When used with `--client`, the `-e` option launches the given command and redirects its standard input, output, and error through the network socket:

```
./athoscat --client -h <server_ip> -p <port> -e /bin/sh
```

This connects to the remote server and gives it interactive access to a shell.

5.2 Security Measures

Because this feature can be dangerous, AthosCat includes built-in security precautions:

- Only the **root user** is allowed to use the `-e` option.
- It is only supported in `--client` mode to prevent blind command injection.
- If used incorrectly, the program will exit with an error and display an appropriate message.

5.3 Listener Confirmation

In `--listener` mode, AthosCat asks for confirmation before opening a listening port unless the `-y` flag is passed:

```
[!] Listening mode selected. Confirm? [y/N]
```

This provides an extra layer of safety to avoid unintended exposure to incoming connections.

Example: Remote Shell

On the listener:

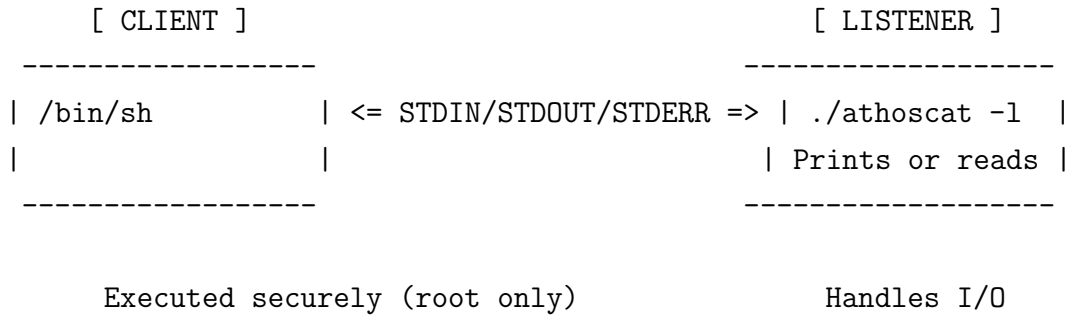
```
./athoscat --listener -p 9001 -v
```

On the client:

```
sudo ./athoscat --client -h 192.168.1.10 -p 9001 -e /bin/sh
```

Result: The listener receives a shell session from the client.

5.4 ASCII Diagram



Use Cases

- Reverse shell for diagnostics in secure LAN environments.
- Remote command chaining with programs like `tar`, `gzip`, `openssl`.
- Controlled automation or scripting over sockets.

5.5 Warning

This option should **never** be used in untrusted or public-facing environments without additional safeguards (e.g., SSH tunnels, VPNs, or firewalls). While AthosCat restricts usage to `root`, it remains a powerful feature that should be used responsibly.

6 Conclusion

This project allowed me to rebuild and extend the core functionalities of a fundamental tool like `netcat`, while integrating additional layers of control and safety. By developing AthosCat, I gained a much deeper understanding of socket programming, protocol handling (TCP/UDP), and the challenges of designing secure and user-friendly network utilities.

Working on this project significantly strengthened my ability to anticipate and mitigate potential security issues, especially when implementing features such as the `-e` option, which can pose serious risks if misused. It was essential to balance functionality with responsible design, and this process gave me valuable insight into secure software practices.

More broadly, recreating a tool I frequently use helped me understand its inner workings and limitations, making me more confident and intentional in my use of it. This hands-on approach not only improved my technical skills but also reinforced what truly drives me in my work: understanding what I use, solving real-world problems, and building reliable tools. It has enhanced both my practical knowledge and my ability to face security challenges with a more informed mindset.