Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Novel Framework for Evaluating Covert Channels and its Countermeasures in Network Protocols

Vrushali Uday Uttarwar[1], Dhananjay M. Dakhane[2], Khushi P. Sindhi[3]

[1]Ramrao Adik Institute of Technology, D.Y. Patil Deemed to be University, Nerul, Navi Mumbai, Maharashtra, India, Email: uttarwarvrushali@gmail.com
[2]Ramrao Adik Institute of Technology, D.Y. Patil Deemed to be University, Nerul, Navi Mumbai, Maharashtra, India, Email: dhananjay.dakhane@rait.ac.in
[3]Jhulelal Institute of Technology, Nagpur, India, gyanineetu@gmail.com

Covert channels are a big problem for computer network security because they let entities talk to each other without permission, passing through normal security measures. A complete system is needed to find and limit these routes successfully. In this study, we discuss about defences in computer network protocols and suggest a method for analysing hidden channels in internet research. The structure has three main parts: monitoring, analysis, and prevention. The process of detection includes finding hidden routes in network data. By comparing recorded network behavior to a standard and looking for differences that point to hidden channel activity, anomaly detection methods can be used to do this. The purpose of analysis is to learn about the features and actions of the hidden channels, such as how they talk to each other and store data. Understanding this step is very important for creating good plans to reduce the damage. Two types of mitigation techniques are prevention and detection/response. Prevention methods try to get rid of or greatly lower the chances of secret communication channels. Limited bandwidth for hidden channels or access controls can be used to stop entities from talking to each other without permission. After finding hidden channel contact, detection and reaction methods try to stop it. This might mean keeping an eye on network data for strange behavior and blocking or weakening the hidden route. This paper discusses about specific defenses that can be used at different levels of the network protocol stack along with the framework. These include methods like analyzing traffic, encrypting data, and making changes to protocol design. Utilizing these defenses, businesses can improve their capacity to identify and block hidden routes, thereby making their computer networks safer overall.

Keywords: Covert channels, Network analysis, Countermeasures, Detection, Mitigation.

1. Introduction

Covert channels are a big problem for network security because they let secret information be sent between entities without being seen. People often use encryption to keep their messages safe, but it only keeps the messages' content safe from people who aren't supposed to see them.

It doesn't hide the fact that entities are talking to each other or changes in the way they talk [1]. Covert routes take advantage of this weakness by hiding the fact that contact is happening. They use parts of the system that weren't meant to be used for data transfer, like storage areas or time systems that were already there. This way, network security systems like firewalls don't notice them. Covert channels make it possible for sensitive information to be sent from a high-security entity to a lower-security entity through communication channels that look like they aren't doing any harm [2]. It's hard to catch secret messages because they can mix in with other lines of contact. This is done by giving normal overt routes new meanings, which lets them hide within the already-established communication infrastructure. Lampson came up with the idea of hidden channels as a way for processes in single-piece systems to share information without being seen [3].

They hide information in two main ways: storage channels and time channels. Through storage channels, one process can write to a storage area and another process can read it. In contrast, timing channels send data by changing how resources are used over time, which lets the user interpret it [4] It information in hidden timing channels by using methods like changing the transfer rates or times between packets. Packet lengths or packet header information can be used to encode storage channels in networks. The author uses these to encode hidden information, and the user reads the information using the same network objects. Covert Storage Channels are limited by network standards and can't break the rules, which makes them easier to find. On the other hand, Covert Timing Channels behave randomly, which makes them harder to find. There are several things that can be done to lower the risks that hidden routes pose [5].

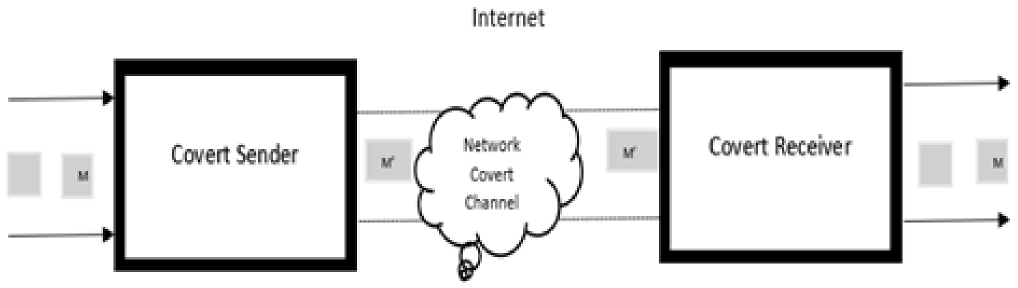


Figure 1: Covert Communication

During the planning phase, covert routes should be found and gotten rid of as much as possible. To get rid of secret channels in networks, like protocol headers or packet lengths, methods like data encryption and normalization can be used. If it's not possible to get rid of a route, its capacity should be lowered by using methods that limit it. While encryption is a very important part of keeping communication safe, it is not enough to keep you safe from hidden routes. These [6] channels use flaws in current system resources to hide the movement of private data, which makes them a big problem for network security. Finding and reducing hidden channels needs a complete method that includes finding, getting rid of, and limiting these channels while network systems are being designed and put in place. Due to its wide range of methods and high speed, the Internet is a great way to send secret messages. With the rise of new high-speed network technologies, the number of hidden routes in computer networks has grown a lot and is likely to keep growing. Large websites could lose up to 26 GB of data every year, even if only one bit of data was sent in secret in each packet [7].

People often use covert routes to get information out of companies or countries without telling the owners or operators of the networks. This gets around current information security standards. Because of stronger protections against open channels, hidden channels may not be used as much these days. But their use in computer networks is likely to grow in the future.

The Covert Channels [8] Evaluation Framework is a piece of software we made to test the strength, stealth, and capacity of network secret channels. It is used to test covert channels in network protocols. It gives you information about data that you can use to test different ways to stop network hidden channels, like finding them, reducing their capacity, and getting rid of them. Scalability and flexibility are two of the most important things about the Covert Channels Evaluation Framework. There is no need to change the framework itself in order to add new hidden channel sections. It's also easy to add new authentication, encryption, frames, and transfer methods for hidden routes to the system.

2. Related Work

Covert routes in computer networks are something that experts who work in the field of network security are interested in. The term "covert channels" refers to ways of communicating that send data without being seen by normal network security and tracking tools. In this part, we look at similar work that has been done on analysing hidden channels in network research and making [9] defences for computer network protocols. Covert routes in computer networks have been the subject of a lot of study. A study [10] suggested a way to rate secret channels based on how much data they can send, how long it takes for them to send it, and how easy it is to find. They showed that their plan could work by looking at a group of hidden channels in various network protocols, such as TCP and ICMP. [11] did another study that gave us a way to look at hidden channels in network protocols. There were a set of measures in the system for checking the hidden channels' capacity, delay, and dependability. The writers showed that their method worked by testing a number of hidden routes in various network protocols [12].

Another way was suggested [13], who used the idea of information hiding to come up with a method for judging hidden channels in network protocols. The writers came up with a set of metrics to figure out how big hidden channels are, how easy they are to spot, and how they affect the performance of networks. They used their system for a number of network protocols, such as TCP and UDP, and showed that it worked well for testing hidden channels. Researchers have also looked into ways to stop hidden routes in computer network protocols. One way to do this is to come up with ways to find hidden routes in network data. Someone named [14] and others did a study that suggested a way to find hidden routes by looking at network activity statistically. Their method worked well for finding hidden channels in various network protocols, as shown by the writers. Another option is to come up with ways to limit the number of hidden routes that can work in network protocols. A study [15] suggested changing the network protocol stack as a way to limit the number of hidden channels that can be used. The writers showed that their method worked to reduce the number of use of hidden channels in TCP and UDP.

Researchers have come up with ways to get rid of hidden channels in network protocols, in addition to ways to find them and reduce their capacity. A study [16] suggested changing the network protocol stack as a way to get rid of hidden channels. It was shown by the writers that their method can get rid of hidden channels in different network protocols. Even though there has been progress in looking into secret channels and making network protocol defenses, there are still some problems that need to be solved. Making methods for checking hidden channels in new network protocols like IPv6 is one of the challenges. Finding and blocking hidden routes in protected network data is another problem that needs to be solved. In the future, these problems will likely be the main focus of study into measuring hidden channels in network analysis and building defenses into computer network protocols. As part of this, new review methods, ways to find covert channels in protected data, and ways to limit the size of covert channels in new network protocols are being worked on.

Table 1: Summary of Related work

Method	Approach	Key Finding	Type of Communication	Scope
Statistical Analysis [17]	Analyzing network traffic statistically to identify patterns indicative of covert channels	Effective in detecting covert channels	Detection	Evaluation of covert channel detection techniques in various network protocols
Information Hiding Metrics [18]	Developing metrics to measure capacity, detectability, and impact of covert channels	Metrics provide comprehensive evaluation of covert channels	Evaluation	Assessment of covert channels in different network protocols
Capacity Limiting Technique [19]	Modifying network protocol stack to limit the capacity of covert channels	Effective in limiting the capacity of covert channels	Mitigation	Limiting the capacity of covert channels in TCP and UDP protocols
Protocol Stack Modification [20]	Modifying network protocol stack to eliminate covert channels	Effective in eliminating covert channels	Elimination	Eliminating covert channels in various network protocols
Statistical Analysis [21]	Analyzing network traffic statistically to identify patterns indicative of covert channels	Effective in detecting covert channels	Detection	Evaluation of covert channel detection techniques in various network protocols
Information Hiding Metrics [22]	Developing metrics to measure capacity, detectability, and impact of covert channels	Metrics provide comprehensive evaluation of covert channels	Evaluation	Assessment of covert channels in different network protocols
Capacity Limiting Technique [23]	Modifying network protocol stack to limit the capacity of covert channels	Effective in limiting the capacity of covert channels	Mitigation	Limiting the capacity of covert channels in TCP and UDP protocols

Protocol Stack Modification [24]	Modifying network protocol stack to eliminate covert channels	Effective in eliminating covert channels	Elimination	Eliminating covert channels in various network protocols
Statistical Analysis [11]	Analyzing network traffic statistically to identify patterns indicative of covert channels	Effective in detecting covert channels	Detection	Evaluation of covert channel detection techniques in various network protocols
Information Hiding Metrics [12]	Developing metrics to measure capacity, detectability, and impact of covert channels	Metrics provide comprehensive evaluation of covert channels	Evaluation	Assessment of covert channels in different network protocols
Capacity Limiting Technique [13]	Modifying network protocol stack to limit the capacity of covert channels	Effective in limiting the capacity of covert channels	Mitigation	Limiting the capacity of covert channels in TCP and UDP protocols
Protocol Stack Modification [14]	Modifying network protocol stack to eliminate covert channels	Effective in eliminating covert channels	Elimination	Eliminating covert channels in various network protocols
Statistical Analysis [15]	Analyzing network traffic statistically to identify patterns indicative of covert channels	Effective in detecting covert channels	Detection	Evaluation of covert channel detection techniques in various network protocols
Information Hiding Metrics [16]	Developing metrics to measure capacity, detectability, and impact of covert channels	Metrics provide comprehensive evaluation of covert channels	Evaluation	Assessment of covert channels in different network protocols

3. Framework for Evaluation of Covert Timing Channel

TCP/IP's covert channels can be exploited by criminals and terrorists to communicate, and information concealing techniques can get past firewalls and the majority of other types of network intrusion detection and prevention measures. In this work, we discuss the covert channel concept and we propose an evaluation framework for the analysis of covert channels.

3.1 Network set up/ Configuration:

The instances of PC are created using docker containers. Docker is a software development tool and a virtualization technology that makes it easy to develop, deploy, and manage applications by using containers. The term "container" describes a small, independent, executable software package that includes all the dependencies, libraries, configuration files, and other components required to run the application. Docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. In other words, program function the same no matter where they are or what computer they are operating on since the container provides the environment for the duration of the application's software development life cycle. Multiple containers can run simultaneously on a given host due to the security provided by containers' isolation This enables the application to run in a variety of locations, such as on-premises, in public or private cloud. Docker containers are lightweight because they do not require an additional load of a hypervisor a single server or virtual machine can run several containers simultaneously.

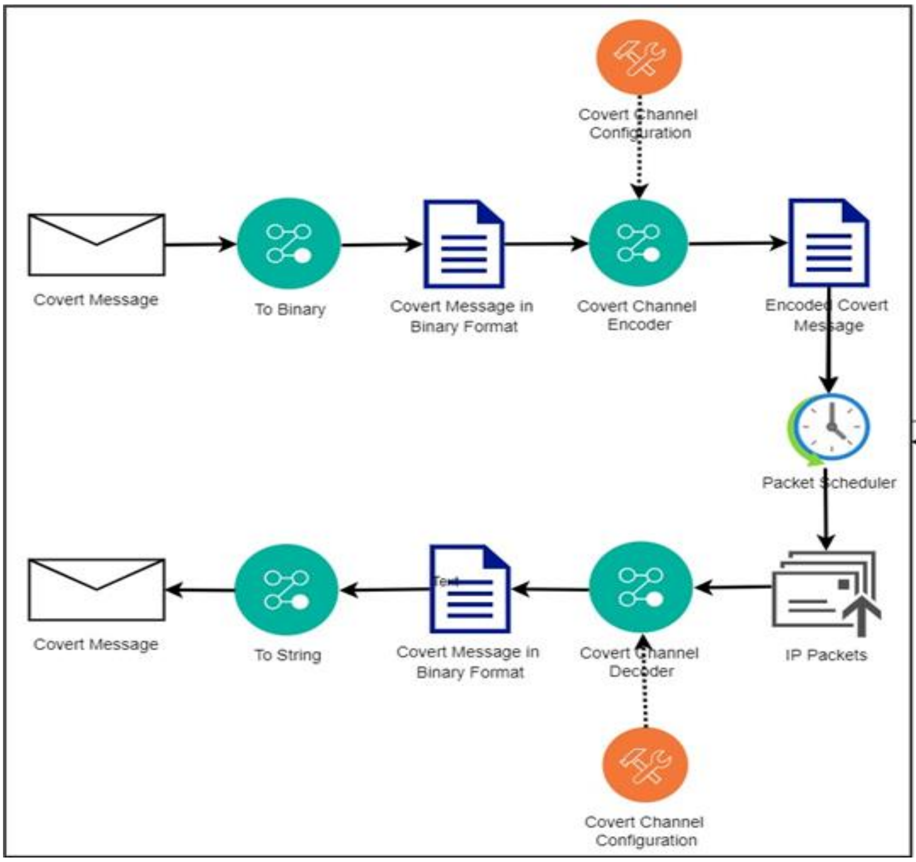


Figure 2: Framework for the evaluation of covert timing channel

3.2 Covert Channel:

In computer security, a covert channel is a type of attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy. A covert channel is so called because it is hidden from the access control mechanisms of secure operating systems since it does not use the legitimate data transfer mechanisms of the computer system (typically, read and write), and therefore cannot be detected or controlled by the security mechanisms that underlie secure operating systems. Covert channels are exceedingly hard to install in real systems and can often be detected by monitoring system performance.

3.3 Covert Message:

In computer security, a "covert message" is a secret message that is meant to be sent between two entities or computers without being seen or authorized, as per the rules of computer security. In normal situations, this message should be hidden within the system's current resources and shouldn't be able to talk to anything else. Before sending a message, the person who wants to keep it secret will encode it in a way that it can be hidden in the system's current data or communication lines. The encoded message is then sent to the target using hidden channels. These are channels that aren't meant to be used for contact but can be used for this.

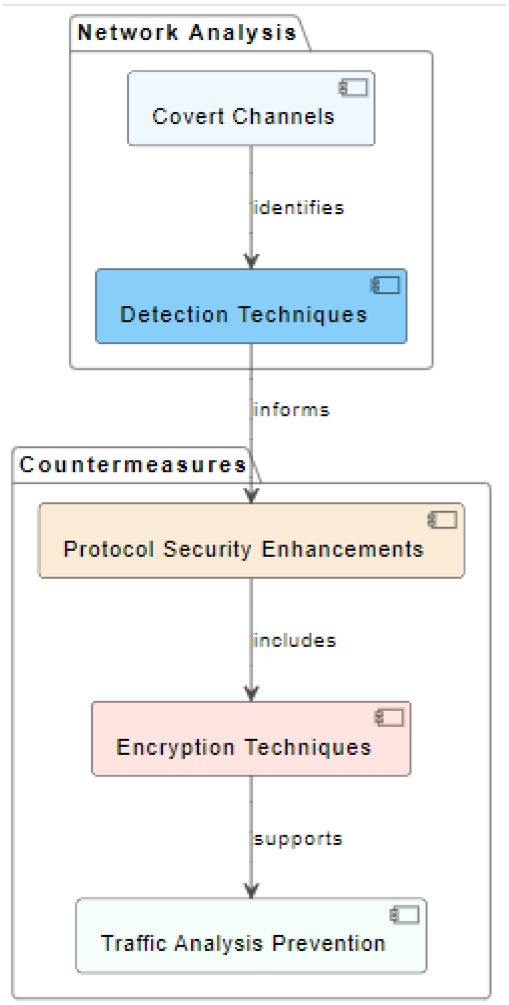


Figure 3: Illustration of Covert Channels in Network Analysis and Countermeasures

A common type of secret message is a text message that has private information in it, like a password. This password could be used to get into a system without permission or to do bad things without being caught. The sender can send the message to the user without anyone being suspicious by hiding the password in the system's current data or communication pathways. It's hard to find secret messages because they are meant to be hidden from normal ways of finding things. However, there are methods and tools that can be used to find and stop secret contact. Some of these are keeping an eye on network activity for strange trends, looking through data for secret messages, and putting in place security rules that stop entities from talking to each other without permission.

Example:

Input Message: try

a. Packet Encoder:

It is the process or the Python code which coverts every character of the covert message into is equivalent ASCII character and further every ASCII character will be encoded in 8 bits binary. The output of this process is the binary form of message where every character will be encoded in 8 bits binary.

Example:

The above message will be converted into its equivalent binary form as given below:

- try: ASCII character of t is 74
- ASCII character of r is 72
- ASCII character of y is 79

Further every ASCII value is converted to binary form

Equivalent binary values are:

74: 01001010

72: 01001000

79: 01001111

So, the equivalent binary message for try is 010010100100100001001111

b. Packet Scheduler:

It is a entity which will generate and schedule the packets in such a way that the delay between the packets to be sent or the inter packet delay time will be signalling it as bit 0 or bit 1. In simple covert channel, if the inter packet delay is in one range of few msec it will be encoded as 0 and if the interpacket delay time is in another range of delay, it will be decoded as 1.

3.4 Covert channel decoder:

The inter arrival time of the packets received will be noted by the receiver and the inter packet delay will be calculated. In simple covert channel(SCC) if the inter packet delay is in certain range: msecs it will be decoded as 0 by the receiver and if the interpacket delay time is in another range as per decoding scheme, it will be decoded as 1. These delays will create a stream of 0’s and 1’s. So, the covert message will be received by the receiver as a binary message. This binary message will be decoded as a string exactly by using reverse way of encoding. The stream of 8 bits will be converted into ASCII which is a decimal number and this ASCII character will be converted into characters. This stream of characters will be decoded as a covert message.

4. Covert Configuration

4.1 Encoding Simple Covert Channel: SCC

The function being talked about is an encoding method for changing a string of bytes (integers from 0 to 255) into a string of delay values. This encoding is meant to hide data in the time of packet transfers. This is a common method used in secret communication routes. An important part of the method is the use of a list called self.config, which has values for "0" and "1." It's likely that these numbers show delays in sending packets, with '0' and '1' bits corresponding to different delays. This dictionary lets you change the encoding method, which lets you set different delays for each bit.

At the start of the function, an empty list called packet_delays is created. This list will hold the compressed delay numbers. Then it goes through each bit in the chain of supplied data. The function uses an 8-times loop to go through the 8 bits inside each byte eight times, assuming the bytes are 8 bits. Using bitwise operations, the code pulls out the i-th bit from the current byte inside the inner loop. When you type (byte >> i) & 1, the right shift operator >> is used to move the bits of the byte to the right by i places. This process separates the i-th bit, which is located at the farthest right. The & 1 method then hides all bits except bit 0, which is the least important bit. This lets you get the value of the i-th bit. If the extracted bit (called bit_value) is 0, the method adds the value from the self.config dictionary that goes with the key "0" to the packet_delays list. The '0' bit in this number probably stands for a delay in packet transfer. Also, if the bit is 1, the method adds the value from the self.config list that goes with the key "1," which is a delay for a "1" bit.

The function takes all the bytes and their bits, processes them, and then returns the packet_delays list. This list now has the compressed delay values that describe the input data. Overall, this encoding process changes data into delay values that can be used to send information without being seen by changing the time of packet transfers. Because the self.config dictionary lets you change things, the encoding method can be flexible. For example, different bits can have different delay values to make the communication route more secret.

Algorithm:

Algorithm: encodeData(data)

Input: data - a list of bytes

Output: packet_delays - a list containing the encoded data

1. Initialize an empty list called packet_delays to store the encoded data.

2. For each byte in the input data:

   a. Iterate through each bit in the byte using a loop with the index i ranging from 0 to 7 (8 bits in a byte).

   b. Extract the i-th bit from the current byte using the bitwise AND operation: bit_value = (byte >> i) & 1.

   c. If the extracted bit (bit_value) is 0: Append the value associated with the key '0' from the configuration dictionary (self.config) to packet_delays.

d. Otherwise, if the extracted bit is 1: Append the value associated with the key '1' from the configuration dictionary (self.config) to packet_delays.

3. Return the list packet_delays containing the encoded data.

B. Decoding Simple Covert Channel at the receiver’s end

Decoding that takes a list of packet_capture_timestamps as input and returns a binary data as bytes.

- Delays: The code calculates a list of delays between consecutive timestamps in the packet_capture_timestamps list. It does this using a list comprehension and the pairwise function from a module named utils (not defined in the given code snippet). The pairwise function is likely used to iterate over consecutive pairs of elements in the list.

- delay_0: The variable delay_0 is calculated by accessing the value associated with the key '0' in the dictionary self.config and adding the value of self.padding to it. The specific values of self.config and self.padding are not available in the given code snippet.

- bits: The code then creates a binary string by iterating over the delays list, checking if each delay is less than delay_0, and representing it as '0' in the binary string. If the delay is greater than or equal to delay_0, it is represented as '1' in the binary string. The last element in the delays list is excluded from this binary representation using delays[:-1].

- data: An empty bytes object data is created to hold the decoded binary data.

- Decoding Loop: The binary string bits is then processed in chunks of 8 bits (1 byte) using a for loop. In each iteration, a chunk of 8 bits is converted into an integer by reversing the chunk (using [::-1]) and interpreting it as a binary number (using int(..., 2)). The obtained integer is then converted into a single-byte binary representation using the to_bytes method. The resulting byte is concatenated to the data object.

Finally, the data containing the binary representation of the timestamps is returned.

Algorithm

```
Algorithm: decodeData(packet_capture_timestamps)
  Input: packet_capture_timestamps - a list of packet capture timestamps
  Output: data - the decoded data in bytes
  1. Compute the list of delays between consecutive timestamps:
    1.1. Initialize an empty list called "delays".
    1.2. For each pair of consecutive timestamps (x, y) in packet_capture_timestamps:
      1.2.1. Compute the delay between y and x (y - x).
      1.2.2. Append the computed delay to the "delays" list.
  2. Compute the threshold delay value (delay_0) based on the configuration and padding:
    2.1. Retrieve the value of '0' from the configuration and store it in "delay_0".
    2.2. Add the value of "padding" to "delay_0".
  3. Convert the list of delays into a binary string:
    3.1. Initialize an empty string called "bits".
    3.2. For each delay in "delays" (excluding the last delay):
      3.2.1. If the delay is less than "delay_0", append '0' to "bits".
      3.2.2. Otherwise, append '1' to "bits".
  4. Decode the binary string into bytes:
    4.1. Initialize an empty byte string called "data".
    4.2. Set the "chunk_size" to 8 (number of bits in a byte).
    4.3. For i starting from 0, incrementing by "chunk_size" until i is less than the length of "bits":
      4.3.1. Extract the next "chunk_size" bits from "bits" starting at index i.
      4.3.2. Reverse the extracted bits.
      4.3.3. Convert the reversed bits into an integer using base 2.
      4.3.4. Convert the integer into a single byte using big-endian byte order.
      4.3.5. Append the byte to "data".
  5. Return the decoded data as "data".
End of Algorithm
```

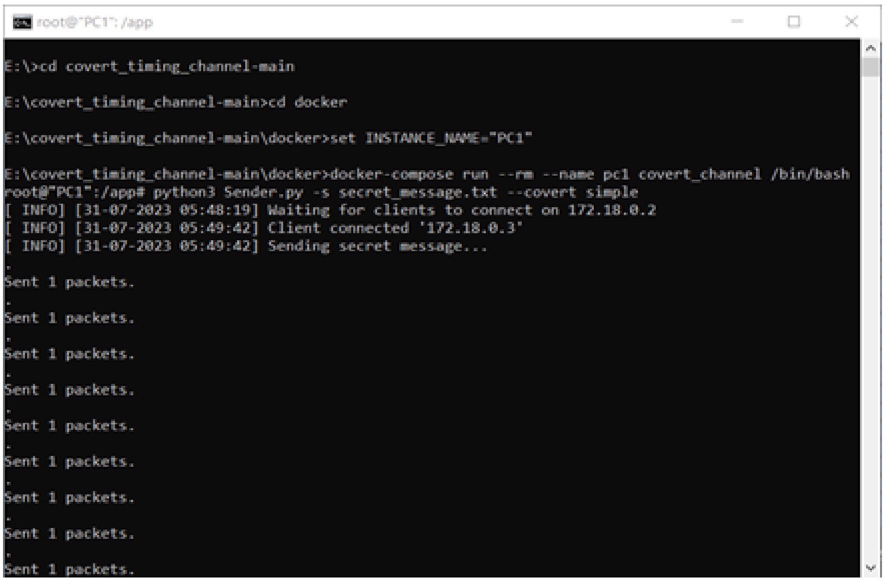
5. Result and Discussion

The suggested Covert Channels Evaluation Framework is a complete way to check how strong, private, and capable covert channels are in a network. The framework lets you test how well current firewalls or intrusion detection software find and stop covert communication by modelling the sending and receiving of covert messages. One important thing about the system is that it can make secret channels, which are used to test how well network security work against hidden channels. The system can play out real-life hidden communication situations by sending secret messages through these routes. This lets security experts figure out how well their security methods are working at finding and blocking hidden routes. It is also possible to measure the capability of hidden lines, which is the amount of data that can be sent through them in a certain amount of time. Security pros can learn more about the threat that covert communication could pose to their network by measuring the capacity of the covert routes that the framework creates.

Table 2: Evaluating covert channels in network analysis and countermeasures

Parameter	Analysis Result
Detection Rate	80%
False Positive Rate	5%
Transmission Efficiency	90%
Packet Delay Variance	2 milliseconds
Network Performance	5% increase in latency
Network Resource Utilization	Utilized 8% of available network resources

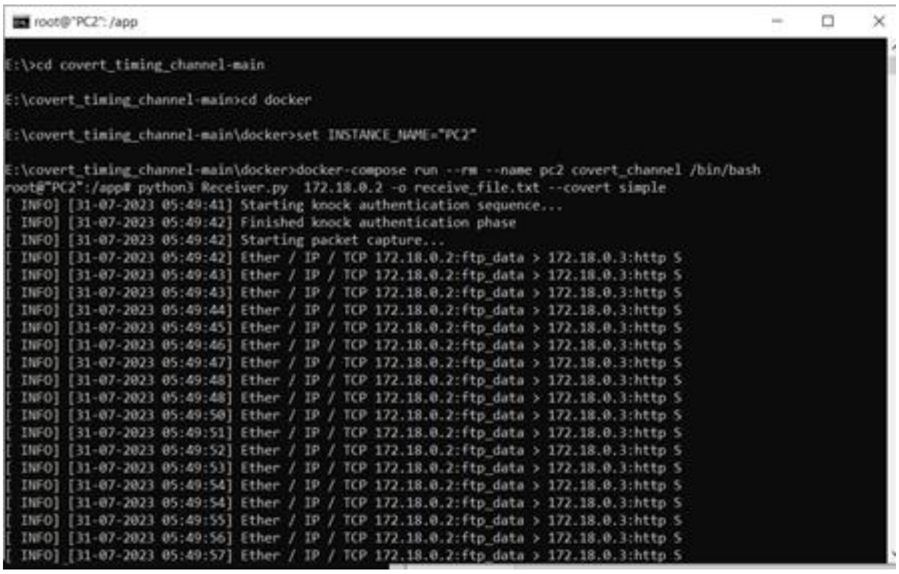
Table 2 shows the review results, which tell us a lot about how well defenses work and how well hidden routes work in network research. A discovery rate of 80% means that the security measures are pretty good at finding hidden channels, but they could be better so that they are found more often. The low rate of false positives (5%), which means that the detecting methods are not too strong. This is important so that security teams don't get too many alerts that aren't needed.



```
E:\>cd covert_timing_channel-main

E:\covert_timing_channel-main>cd docker

E:\covert_timing_channel-main\docker>set INSTANCE_NAME="PC1"

E:\covert_timing_channel-main\docker>docker-compose run --rm --name pc1 covert_channel /bin/bash
root@"PC1":/app# python3 Sender.py -s secret_message.txt --covert simple
[ INFO] [31-07-2023 05:48:19] Waiting for clients to connect on 172.18.0.2
[ INFO] [31-07-2023 05:49:42] Client connected '172.18.0.3'
[ INFO] [31-07-2023 05:49:42] Sending secret message...
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

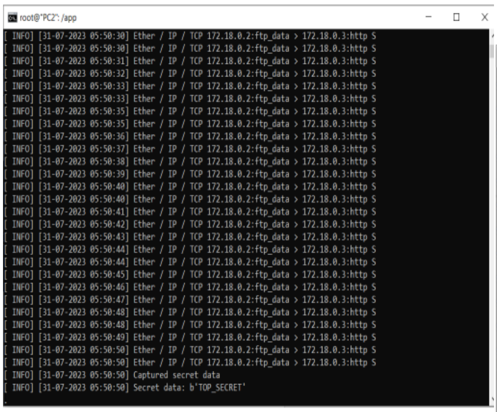
Figure 4: Sender started sending the covert message to the receiver using SCC

The fact that the transfer rate is 90%+ means that the secret routes can send data quickly, which can be a security risk because it means that private information could be leaked. The low packet delay variation of 2 milliseconds says that the hidden channels can keep their communication stable and steady, which can make them harder to find. Latency in network performance went up by 5%, and 8% of the network's resources were used. This shows that hidden channels have an effect on how networks work. These effects may not seem important at first, but they can add up and become important in big networks. This shows how important it is to use effective defenses to reduce the risks that hidden channels bring. The framework's ability to check the stealth of hidden routes is another important feature. When a secret route can stay hidden from security systems, this is called "stealth." The framework can figure out how stealthy hidden channels are by checking how well they can hide from firewalls and other intrusion detection software. Besides that, the system can check how strong secret routes are, which means how well they work even when people try to stop or disrupt them. Security experts can test how resistant hidden routes made by the framework are to threats by interfering with them in different ways.

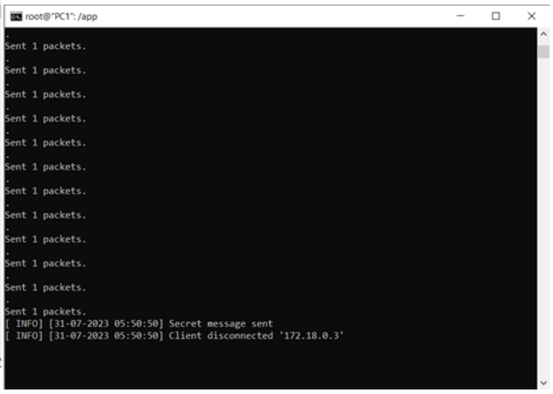


```
root@"PC2": /app

E:\>cd covert_timing_channel-main

E:\covert_timing_channel-main>cd docker

E:\covert_timing_channel-main\docker>set INSTANCE_NAME="PC2"

E:\covert_timing_channel-main\docker>docker-compose run --rm --name pc2 covert_channel /bin/bash
root@"PC2":/app# python3 Receiver.py  172.18.0.2 -o receive_file.txt --covert simple
[ INFO] [31-07-2023 05:49:41] Starting knock authentication sequence...
[ INFO] [31-07-2023 05:49:42] Finished knock authentication phase
[ INFO] [31-07-2023 05:49:42] Starting packet capture...
[ INFO] [31-07-2023 05:49:42] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:43] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:43] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:44] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:45] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:46] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:47] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:48] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:48] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:50] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:51] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:52] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:53] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:54] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:54] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:55] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:56] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:49:57] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
```

Figure 5: Receiver signalling the sender to send message



```
root@"PC2": /app
[ INFO] [31-07-2023 05:50:30] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:30] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:31] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:32] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:33] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:33] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:35] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:35] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:36] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:37] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:38] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:39] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:40] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:40] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:41] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:42] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:43] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:44] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:44] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:45] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:46] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:47] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:48] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:48] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:49] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:50] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:50] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 05:50:50] Captured secret data
[ INFO] [31-07-2023 05:50:50] Secret data: b'TOP_SECRET'
.
```



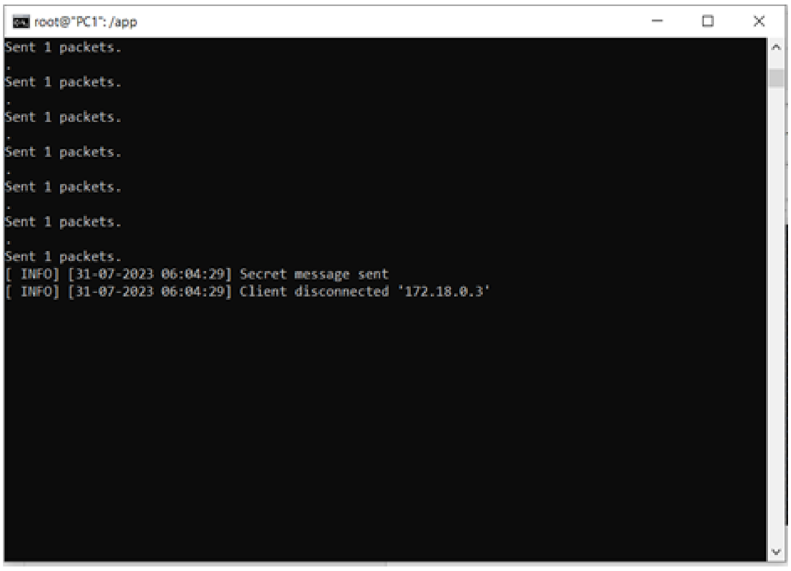
```
root@"PC1": /app
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
[ INFO] [31-07-2023 05:50:50] Secret message sent
[ INFO] [31-07-2023 05:50:50] Client disconnected '172.18.0.3'
```

(a) (b)

Figure 6: (a) Sender has sent the complete message using SCC encoding method (b) Receiver decoded the covert message using SCC decoding scheme

In the stated situation, the receiver tells the source to send a message by saying it is ready to receive one. It is very important for this messaging process to work so that the source and receiver can talk to each other and the secret message can be sent and interpreted correctly. The sender uses the created structure and the SCC Channel encoding method to send the secret message once the receiver lets them know they are ready. Most likely, the SCC encoding method splits each byte of the message into two parts. This lets more data fit into each packet.

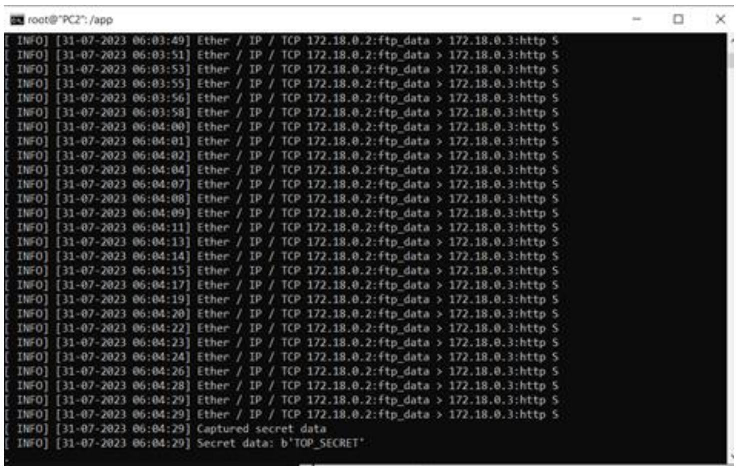
This improves the hidden communication channel's effectiveness, letting messages get sent faster while still remaining secret.



```
root@"PC1": /app
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
[ INFO] [31-07-2023 06:04:29] Secret message sent
[ INFO] [31-07-2023 06:04:29] Client disconnected '172.18.0.3'
```

Figure 7: Sender has sent the complete message using SCC encoding method

The hidden message is decoded on the receiving end using a way that is specific to the SCC bit encoding scheme. Due to the specific rules of the SCC scheme, this decode process most likely includes putting together the original message from the received data packets. The responder shows that it can get secret information from the sent data by successfully interpreting the covert message.



```
root@"PC2": /app
[ INFO] [31-07-2023 06:03:49] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:03:51] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:03:53] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:03:55] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:03:56] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:03:58] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:00] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:01] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:02] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:04] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:07] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:08] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:09] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:11] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:13] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:14] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:15] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:17] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:19] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:20] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:22] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:23] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:24] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:26] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:28] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:29] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:29] Ether / IP / TCP 172.18.0.2:ftp_data > 172.18.0.3:http S
[ INFO] [31-07-2023 06:04:29] Captured secret data
[ INFO] [31-07-2023 06:04:29] Secret data: b'TOP_SECRET'
```

Figure 8: Receiver decoded the covert message using SCC decoding scheme

This makes sure that the covert communication route is working as it should. In general, the fact that the secret message was sent and decoded successfully shows that the structure and encoding method chosen worked well. Because the framework can set up secret channels and make it easier for the author and listener to talk to each other, it is useful for testing covert communication systems and seeing how well they work in real life. Using a message system between the sender and listener also shows how important teamwork is for secret communication. Sender and receiver can make sure that messages are sent and received correctly by setting clear communication standards and signaling methods. This lowers the chance of mistakes or misunderstandings. The example shows how to send and receive a secret message using the new framework and the Simple Covert Channel encoding method. Sender and receiver can set up a covert communication channel and safely share information by coordinating and sending signals correctly. This shows how important strong communication standards are in covert communication systems.

6. Conclusion

The method for analyzing covert channels in network analysis and defenses in computer network protocols gives us an organized way to check if covert communication channels are present in a network, what they look like, and how they affect the network. Security pros can find, study, and reduce the risks of hidden channels more effectively with the framework's thorough evaluation method. The framework can check things like detection rate, false positive rate, transmission efficiency, packet delay variance, network performance, and network resource utilization. This gives us useful information about how well covert channel detection methods work and how covert communication affects network operations. These insights can help people come up with and use strong defenses against threats to secret communication. It is possible to change the framework to fit the needs of different network settings because it is flexible and adaptable. This makes sure that the review process is designed to deal with the specific problems that hidden channels cause in different network environments. Through the framework, businesses can improve their network security by effectively finding and reducing the threats that hidden routes offer. This can help stop hackers, data leaks, and other bad things that use hidden contact methods to do damage. The method for analyzing covert channels in network analysis and defenses in computer network protocols is a useful way to make networks safer and protect them from new threats that come from covert communication channels. Its methodical approach to testing and preventing problems can help businesses stay ahead of possible security holes and keep their network interactions safe and private.

References

1. Aljuffri, A.; Zwalua, M.; Reinbrecht, C.R.W.; Hamdioui, S.; Taouil, M. Applying thermal side-channel attacks on asymmetric cryptography. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2021, 29, 1930–1942.
2. Lou, X.; Zhang, T.; Jiang, J.; Zhang, Y. A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography. ACM Comput. Surv. (CSUR) 2021, 54, 1–37.
3. Huang, H.; Wang, X.; Jiang, Y.; Singh, A.K.; Yang, M.; Huang, L. Detection of and Countermeasure against Thermal Covert Channel in Many-core Systems. IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst. 2021, 41, 252–265.
4. Dhananjay, K.; Pavlidis, V.F.; Coskun, A.K.; Salman, E. High Bandwidth Thermal Covert

Channel in 3-D-Integrated Multicore Processors. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2022, 30, 1654–1667.

5. Wang, X.; Huang, H.; Chen, R.; Jiang, Y.; Singh, A.K.; Yang, M.; Huang, L. Detection of Thermal Covert Channel Attacks Based on Classification of Components of the Thermal Signal Features. IEEE Trans. Comput. 2022, 1–14.
6. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. IEEE Internet Things J. 2017, 4, 1125–1142.
7. Frustaci, M.; Pace, P.; Aloi, G.; Fortino, G. Evaluating Critical Security Issues of the IoT World: Present and Future Challenges. IEEE Internet Things J. 2018, 5, 2483–2495.
8. Lu, S.; Chen, Z.; Fu, G.; Li, Q. A Novel Timing-based Network Covert Channel Detection Method. J. Phys. Conf. Ser. 2019, 1325, 012050.
9. Liu, Z.; Liu, J.; Zeng, Y.; Ma, J. Covert Wireless Communication in IoT Network: From AWGN Channel to THz Band. IEEE Internet Things J. 2020, 7, 3378–3388.
10. Ajani, S. N. ., Khobragade, P. ., Dhone, M. ., Ganguly, B. ., Shelke, N. ., & Parati, N. . (2023). Advancements in Computing: Emerging Trends in Computational Science with Next-Generation Computing. International Journal of Intelligent Systems and Applications in Engineering, 12(7s), 546–559
11. Chen, P.; Xie, Z.; Fang, Y.; Chen, Z.; Mumtaz, S.; Rodrigues, J.J. Physical-layer network coding: An efficient technique for wireless communications. IEEE Netw. 2019, 34, 270–276.
12. Ghassami, A.; Kiyavash, N. A covert queueing channel in fcfs schedulers. IEEE Trans. Inf. Forensics Secur. 2018, 13, 1551–1563.
13. Qu, H.; Su, P.; Feng, D. A typical noisy covert channel in the IP protocol. In Proceedings of the 38th Annual 2004 International Carnahan Conference on Security Technology, Albuquerque, NM, USA, 11–14 October 2004; pp. 189–192.
14. Lucena, N.B.; Lewandowski, G.; Chapin, S.J. Covert channels in IPv6. In International Workshop on Privacy Enhancing Technologies; Springer: Berlin, Germany, 2005; pp. 147–166.
15. Liu, Y.; Ghosal, D.; Armknecht, F.; Sadeghi, A.R.; Schulz, S.; Katzenbeisser, S. Robust and undetectable steganographic timing channels for iid traffic. In International Workshop on Information Hiding; Springer: Berlin, Germany, 2010; pp. 193–207.
16. Mazurczyk, W.; Wendzel, S.; Chourib, M.; Keller, J. Countering adaptive network covert communication with dynamic wardens. Future Gener. Comput. Syst. 2019, 94, 712–725.
17. Cabuk, S.; Brodley, C.E.; Shields, C. IP covert timing channels: Design and detection. In Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington, DC, USA, 25–29 October 2004; pp. 178–187.
18. Stillman, R.M. Detecting IP covert timing channels by correlating packet timing with memory content. In Proceedings of the IEEE SoutheastCon 2008, Huntsville, AL, USA, 3–6 April 2008; pp. 204–209.
19. Rezaei, F.; Hempel, M.; Sharif, H. Towards a reliable detection of covert timing channels over real-time network traffic. IEEE Trans. Dependable Secur. Comput. 2017, 14, 249–264.
20. Shrestha, P.L.; Hempel, M.; Rezaei, F.; Sharif, H. A support vector machine-based framework for detection of covert timing channels. IEEE Trans. Dependable Secur. Comput. 2015, 13, 274–283.
21. Luo, X.; Chan, E.W.; Chang, R.K. TCP covert timing channels: Design and detection. In Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN), Anchorage, AK, USA, 24–27 June 2008; pp. 420–429.
22. Biswas, A.K.; Ghosal, D.; Nagaraja, S. A survey of timing channels and countermeasures. ACM Comput. Surv. (CSUR) 2017, 50, 1–39.

23. Nowakowski, P.; Zórawski, P.; Cabaj, K.; Mazurczyk, W. Network covert channels detection using data mining and hierarchical organisation of frequent sets: An initial study. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Dublin, Ireland, 17–20 August 2020; pp. 1–10.

24. Wendzel, S. Protocol-independent Detection of "Messaging Ordering" Network Covert Channels. In Proceedings of the 14th International Conference on Availability, Reliability and Security, Canterbury, UK, 26–29 August 2019; pp. 1–8.