Received 1 January 2024, accepted 12 February 2024, date of publication 23 February 2024, date of current version 13 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3369608

RESEARCH ARTICLE

Creating and Developing a High-Throughput Covert Channel via Program Execution

ABDULRAHMAN ALHELAL AND MOHAMMAD AL-KHATIB
Computer Science Department, Imam Mohammad Ibn Saud Islamic University, Riyadh 11652, Saudi Arabia

Corresponding author: Abdulrahman Alhelal (429000816@sm.imamu.edu.sa)

This work was supported by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University through the Graduate Students Research Support Program.

ABSTRACT A covert communication channel facilitates direct data transfer between two parties through pre-communication knowledge agreements, ensuring secure and confidential transmission of information. However, the existing covert channels suffer from performance limitations, specifically in terms of throughput and speed. The encoding techniques employed in covert channels can be time-consuming and have limited data transfer capabilities. Furthermore, the ability of covert channels to handle files with different formats has not been sufficiently explored. This paper introduces a high-performance implementation of a covert channel that leverages Java exception handling during program execution. To optimize the covert channel's performance, this research explores the use of several encoding methods, including ASCII, Byte, Hexadecimal, Base64, and Huffman coding. The proposed covert channel's performance is evaluated and analyzed for various coding methods. To study the impact on performance, multiple file formats, including text, audio, and video, were used in the experiment. Experimental results showed that the hexadecimal coding method improves the throughput and decreases the time delay of the covert channel. This is attributed to its ability to minimize the number of tries before encountering an ''Index Out of Bounds'' exception. On the contrary, the Base64 method is found to be inefficient as it produces longer strings than the original inputs, resulting in increased time delays during data transfer. The best results are achieved when applying the hexadecimal method with Huffman coding. It takes 6241 milliseconds to transmit a 12.8-megabyte text file, with a throughput of 23116 bits per millisecond.

INDEX TERMS Covert channel throughput, time, optimization, index out of bound, exception, ASCII, byte, hexadecimal, Base64, Huffman coding.

I. INTRODUCTION

A covert channel is a communication channel that allows two entities to indirectly transfer data by specifying and sharing knowledge before communication occurs. Various types of covert channels have been used in encoding, decoding, and transferring original messages [1], [2], [3], [4]. A covert channel is considered a mechanism for violating the communication security policy, which is not anticipated by the system creator [1]. The study [5] presents two covert channels that exploit nonce-based network authentication. The first channel exploits key-based authentication, while the second channel exploits hash-based authentication. These channels are utilized for sending encrypted information between parties and involve challenge-response authentication with a nonce for transferring secret information. The researchers evaluate the throughput performance of their covert channel by measuring the number of attempts required to achieve the challenge-response authentication mechanism. However, most previous work on developing covert channel mechanisms still suffers from low throughput performance when sending files. Additionally, these mechanisms require an effective encoding strategy to enhance data transfer [5], [6].

There are several coding techniques used to compress data and improve the performance of data transfer. In our

The associate editor coordinating the review of this manuscript and approving it for publication was Tiago Cruz.

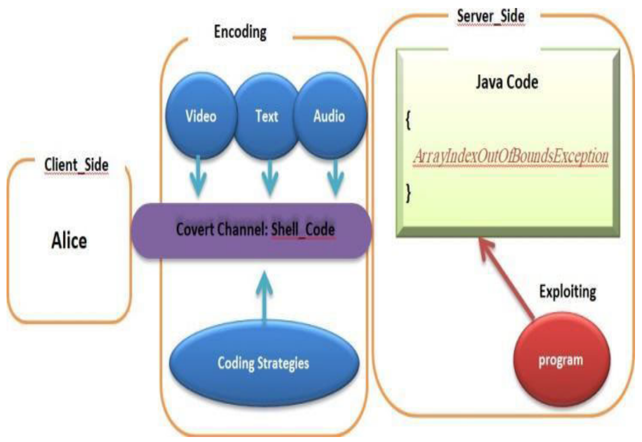


FIGURE 1. The proposed architecture of a covert channel for file transfer.

paper, we utilize Huffman code, Base64 coding, byte coding, ASCII coding, hexadecimal coding, and combinations thereof. ASCII coding [7] is based on a character encoding system that follows the order of the English alphabet. With 128 specified characters, ASCII is typically predicted to be 8 bits long. However, statistical studies on text demonstrate that alternative letter pairings and their frequency can be employed to compress data effectively. Huffman coding [8] is a widely used frequency-dependent lossless compression technique. It allocates shorter code words to frequently occurring letters and longer ones to less frequently occurring ones, thus facilitating data compression. Base64 coding [9] employs a 65-character subset of US-ASCII, allowing for the representation of 6 bits per printed character. An array of 64 printable characters serves as an index for each 6-bit group, with the referred character being added to the output string. Hexadecimal coding [10] is a numbering system with a base value of 16. It uses 16 symbols to represent hexadecimal integers, including values from 0 to 9 and letters A to F.

We develop a new covert channel that exploits Java's "Index Out of Bounds" exception handling. Figure 1 illustrates the steps of the covert channel architecture, which comprises three major components: the client side, encoding, and server side. The covert channel is implemented using a Java program that exploits the "Index Out of Bounds" exception. Our covert channel is designed for transferring files (text, audio, and video) using various coding techniques and encoding methods, including ASCII, Byte, Hexadecimal, Base64, and Huffman coding.

The rest of the paper is organized as follows: In Section II, we provide a comprehensive review of related works about covert channels and their utilization in data transfer. Section III outlines our technique for the development and implementation of covert channels. In Section IV, we present the experimental results and provide a detailed discussion of the findings. Finally, in Section V, we summarize the key findings and draw conclusions based on our research.

II. RELATED WORKS

A. RELEVANT STUDIES

In this section, we focus on the relevant studies that explore covert channels used for data transfer. One study [11] conducts a comprehensive survey on covert channels employed to conceal information within network protocols. They analyze approximately 109 techniques targeting covert communication protocols and classify them based on special patterns. These patterns include size modulation, encoding hidden information in the sequence of header/PDU elements, adding redundancy, PDU corruption/loss pattern, random value pattern, value modulation pattern, reserved/unused pattern (a reserved or unused header/PDU element is used by the covert channel to encode data), inter-arrival time pattern, data rate of traffic, PDU order pattern, and retransmission pattern.

The study [12] presents a survey on the development of network covert channels, covering items such as adversary scenarios, covert channel techniques, and countermeasures. Covert channels are used to transmit information through network protocols. The survey highlights various covert channel techniques, including the use of unused header bits, header extensions, padding, the IP Identifier and Fragment Offset, TCP Initial Sequence Number (ISN), checksum fields, the time to live (TTL) field, modulation of address fields and packet lengths, modulation of timestamp fields, packet rate and timing, message sequence timing, packet loss and packet sorting, frame collisions, and ad hoc routing protocols. Countermeasure techniques to detect and prevent covert channels are also discussed, including channel elimination through host security, network security, and traffic normalization, as well as bandwidth limitation.

Study [2] develops a storage covert channel by utilizing a storage memory as a pre-agreement data store between entities. Data structures are employed to establish the covert channel, enabling secure data transfer between the entities. This mechanism allows parties to agree on a secret key and is flexible in generating keys of different sizes. However, it incurs additional traffic overhead and is time-consuming. Studies [3], [6] utilize stack-overflow attacks and address space layout randomization on Linux to transmit different file formats. The sender attempts to guess the delta_mmap memory (16 bits) while one entity indirectly sends files to another entity through a server located on the same machine. The client guesses the random memory offset of the vulnerable server's standard C library (delta_mmap), and Bob monitors Alice's guessing attempts by observing the server process. The information to be transmitted is encoded in the number of failed guesses before success. Text and audio files are successfully transmitted between the entities with the best throughput performance achieved using hexadecimal coding, as shown in Table 1.

One study [13] explores the utilization of VoIP communications as a technique to enhance privacy in file transfers. They propose the idea of concealing traffic within VoIP conversations to prevent detection and interference. The

TABLE 1. Covert channel performance during transferring audio files [3].

Audio File Size	Optimization Time	Without Optimization
2.3 Kb	.76 Minute	1.01 Minute
1.2 Mb	91. 9 Minute	234 Minute

researchers utilize voice activity detection features available in client interfaces to generate fake silent packets that can serve as carriers for transferring confidential data. The results indicate that this approach could effectively enforce privacy in practical use cases, particularly in file transfers. The researchers also develop an interface for tunneling protocols in the TCP/IP suite, as shown in Table 2. However, it was found that Privacy Enhancing Technology Voice Activity Detection (PETVAD) is slower compared to direct access due to the subpar performance of TCP in the presence of significant delays. The results presented in [13] demonstrate that sending a 1 MB web page incurs a throughput of 283 KB/s and takes 3.61 seconds with a 100 ms delay. The throughput is measured by constructing local area and wide area network configurations, and the researchers use three web pages with varying numbers of inline objects (1, 10, and 100) with sizes of 1 MB, 100 KB, and 10 KB, respectively.

B. RELATED WORKS ANALYSIS

This section focuses on the analysis of related works that explore covert channels utilized for data transfer between entities. Various methods are employed to establish covert channels, enabling the encoding and decoding of information including text files, audio, video, and web pages. The performance of the covert channel is evaluated based on its throughput, which refers to the number of transmitted bits per second. In the reviewed literature, it is observed that covert channels generally exhibit low throughput. To address this limitation, researchers have proposed different mechanisms and enhancements to improve performance, such as using encoding techniques. For a more comprehensive understanding of these mechanisms, refer to [10].

III. CREATING AND DEVELOPING COVERT CHANNEL BASED ON JAVA PROGRAM

To construct and implement our covert channel, we utilize a Java program, as described in Section I. The objective is to establish an indirect interaction between the client and the server computer using the covert channel. In some computer languages, like Java, buffers have a predefined capacity and do not accept data that exceeds their size. To handle this situation, an exception called "java.lang.ArrayIndex Out Of Bounds Exception" is created, which prevents buffer overflow when a user attempts to input data that exceeds the array's size. The code snippet below provides an example of the Java "IndexOutOfBoundsException" exception handling

$$int\ a[] = \{2,\ 3,\ 5,\ 1,\ 20\};$$

TABLE 2. Time and throughput for direct and PETVAD covert channels [10].

web pages size	Time (second)	Rate (KB/s)	Techniques	Delay
1* 1MB	3.61	283	Direct	100 ms
10* 100K	4.98	205		
100 * 10 K	21.65	47		

$$for(int\ i = 2;\ i <= a.length;\ i++)$$
$$System.out.print(a[i]);$$

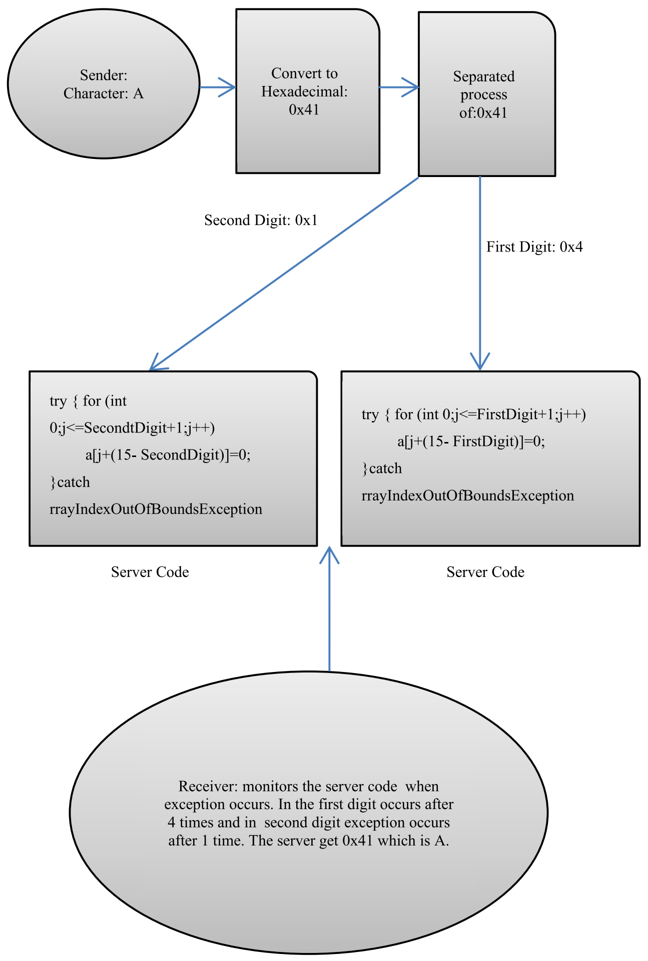
When a user executes the code segment, the output is "2 3 5 1 20" followed by "Exception:java.lang.ArrayIndexOutOf Bounds". We can exploit this exception to create and develop a covert channel. The server program keeps track of the number of buffer accessing operations required until the "Index Out of Bound exception" occurs, allowing us to monitor when this exception happens. If the sender wants to send number 3 to the server program, they simply set the variable "i" to 2. This will cause the function "system.out.print" to be executed three times. The backdoor keeps track of the server software to determine the number of system calls that are made before exception handling occurs. Once a covert channel is established, it can be used to send data in different formats. The channel's performance is evaluated in terms of throughput and time. Various encoding techniques are implemented to assess their impact on the channel's performance. By conducting these implementation scenarios and evaluating the performance using different encoding techniques, the aim is to determine the optimal combination of techniques that achieves the highest performance level for the covert channel. The channel will be utilized in various scenarios, including:

1. Alice will send audio files with different sizes and measure the throughput of the channel using Huffman code, Base64, Byte, ASCII, hexadecimal coding, and combinations of these techniques.

2. Alice will send video files of different sizes and measure the throughput of the channel using Huffman code, Base64, Byte, ASCII, hexadecimal coding, and combinations of these techniques.

3. Alice will send text files of different sizes and measure the throughput of the channel using Huffman code, Base64, Byte, ASCII, hexadecimal coding, and combinations of these techniques.

Figure 2 illustrates a real-life scenario of sending covert data. For instance, if the sender transmits the character 'A' to the receiver, the covert 'A' to the hexadecimal value 0×41. The sender splits the 0×41 into two numbers: '4' and '1', and sends the numbers individually to the server, which. Records the timing of the exception handling.


Sender: Character: A
Convert to Hexadecimal: 0x41
Separated process of:0x41
Second Digit: 0x1
First Digit: 0x4
Server Code
Server Code
Receiver: monitors the server code when exception occurs. In the first digit occurs after 4 times and in second digit exception occurs after 1 time. The server get 0x41 which is A.

```
try { for (int
0;j<=SecondtDigit+1;j++)
        a[j+(15- SecondDigit)]=0;
}catch
rrayIndexOutOfBoundsException
```

```
try { for (int 0;j<=FirstDigit+1;j++)
        a[j+(15- FirstDigit)]=0;
}catch
rrayIndexOutOfBoundsException
```

FIGURE 2. The real-life scenario of sending data over the channel.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. EXPERIMENTS ENVIRONMENT

The experiments for this paper are conducted using Java, which is considered the software for implementing our covert channel. We develop three programs: the client, server, and backdoor. The client establishes a covert channel by exploiting server software to transmit different file formats to a backdoor. Proof-of-concept techniques are utilized to implement our covert channel and measure network throughput and other metrics. We conducted the experiments on a laptop with the following specifications: Intel® CoreTM i3-500u processor, 250GB SSD hard disk, 4GB RAM, and the Windows 10 operating system. To manage and organize the experimental findings, we employ Excel software. We compute the average results and create graphs to represent time and throughput. These graphs aid in analyzing and interpreting the experimental data. In addition to Java, we use programs such as Excel and Word for the creation and evaluation of our covert channel. Excel software is utilized for computing the average results and creating graphs to represent time and throughput, facilitating the management and organization of the experimental findings.

TABLE 3. Applying several optimization techniques to our covert channel.

File type	Size	optimization				Performance
		Byte coding	Hex coding	Huffman+ hexadecimal	Base64 code	
Text	1(KB)- 12.4(MB	yes	Yes	yes	Yes	Time And Throughput
Audio (WAV)	1 (MB)-29 (MB)	Yes	Yes	NO	Yes	
Video (MP4)	(1.5)MB-18(MB)	NO	Yes	No	Yes	

As represented in Table 3, the parameters of how the client sends files to a backdoor through an exploited server program. Firstly, different file formats, such as text, audio, and video, are employed. Secondly, different optimization techniques, including byte, hexadecimal, Huffman, and base64 coding, are utilized. Thirdly, optimization techniques are combined. Fourthly, the performance of our covert channels is measured in terms of time and throughput. To enhance time and throughput performance, we combine optimization strategies. Regardless of the optimization methods used, the sender transmits files to the server, and the time and throughput required for the backdoor to receive the files are tracked. Each experiment is repeated five times before averaging the results. Free audio, video, and text files of different sizes are downloaded from free websites. In this study, throughput refers to the number of bits sent via the covert channel per second. The time it takes for data to be transmitted through our covert channel is the subject of our paper, referred to as ‘time’ [3], [6], [13], [14].

We select several text files of varying sizes, along with audio files in WAV format and video files in MP4 format. These files are used in our experiments to evaluate the performance of our covert channel.

B. EXPERIMENT RESULT

This research introduces comprehensive experiments to evaluate proposed covert channel implementations and find out their performance results. As mentioned in the previous section of this research, experiments include implementing covert channels with a variety of encoding methods. Moreover, experiments involve implementing a covert channel with a combination of two or more encoding methods. This aims to find out the best optimization strategy that boosts the throughput of the channel and reduces the time delay for sending data.

The experimental results are depicted in Figures 3–15. These figures mainly showcase the correlation between file size (in KB or MB) and time consumption (in milliseconds). As the file size increases, time consumption also increases, leading to degradation in the performance of the covert channel. To address this issue, encoding techniques are employed.

When employing byte optimization to send text files between two entities, it takes approximately 138 seconds to transmit a 16.6 MB text file. The time duration is considered significantly high when compared to other widely-used

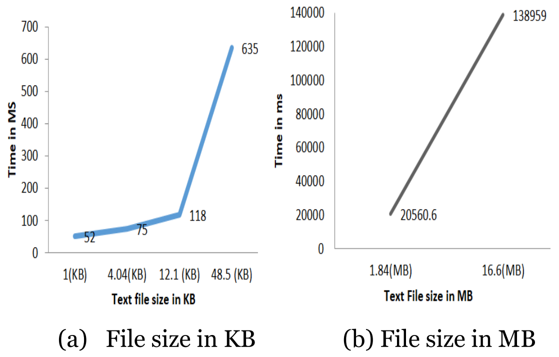


FIGURE 3. A graph showing the speed of file transfer using the covert channel method.

optimization strategies, as shown in Figure 3. In Figure 3 (a), the text file sizes are represented in KB, while in Figure 3 (b), they are represented in MB. The experiments are conducted using both smaller (KB) and larger (MB) file sizes to assess the behavior of the proposed method concerning the file size. The experimental results demonstrate that there is no significant impact on performance when the size of files is below 12.1KB.

Figure 4 illustrates the relationship between file size (in MB) on the horizontal axis and throughput (bits/sec) on the vertical axes. As the file size increases, the throughput performance decreases. Specifically, when employing byte optimization, it requires approximately 1000 bits per second to transmit a 16.6 MB text file. This throughput rate is notably lower compared to other commonly utilized optimization strategies.

The relationship between text file size (in MB) on the horizontal axis and time (in milliseconds) on the vertical axes is shown in Figure 5. In this experiment, we used hexadecimal optimization. As the file size grows, the time performance is significantly improved compared with byte-coding optimization. Using hex-optimization optimization, it takes approximately 10 seconds to send a 16.6 MB text file. Which, in comparison to other optimization strategies, is considered to have significantly high performance.

Hexadecimal and byte coding for sending text files are evaluated in terms of time and throughput, as depicted in Figures 6 and 7, respectively. When transferring a 16.6 MB file, the time required using hexadecimal coding is 9679 ms, whereas, with byte coding, it took 138959 milliseconds.

In Figure 7, the throughput performance is compared between the byte-coding and hexadecimal-coding methods. Hexadecimal and byte coding for sending audio files are evaluated in terms of time and throughput, as shown in Figures 8 and 9, respectively. When transferring a 10 MB audio file, the time required using hexadecimal coding is 1.6 seconds, while with byte coding it takes 4.7 seconds.

Figure 9. An analysis of the efficiency of sending audio files using hexadecimal and byte optimization in terms of throughput.

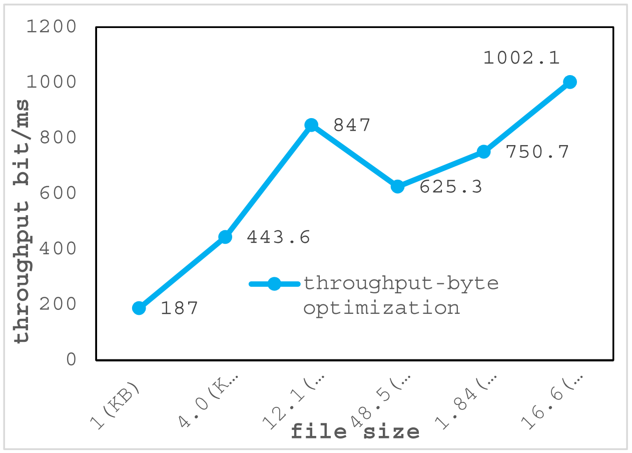


FIGURE 4. A graph showing the relationship between file size and throughput.

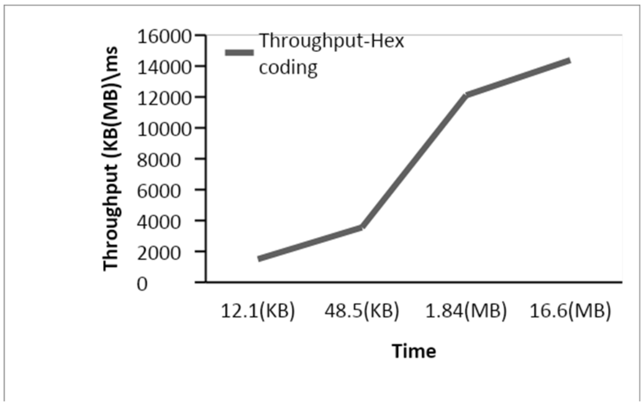


FIGURE 5. Displays a graph illustrating the throughput of file transfers using hexadecimal optimization.

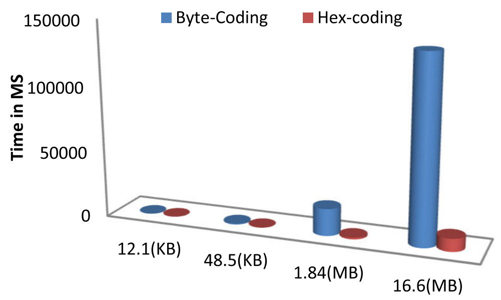


FIGURE 6. A graph comparing the file transfer rates between Byte-coding and Hexadecimal-coding methods.

Figure 10 illustrates that, for both video and audio formats, hexadecimal optimization exhibits superior time performance compared to byte optimization during file transfer.

Figure 11 demonstrates that, for both video and audio formats, hexadecimal optimization outperforms byte optimization in terms of throughput performance while transferring files.

Figure 12 demonstrates that when using video format, hexadecimal optimization achieves better throughput performance compared to base-64 optimization during file transfer.

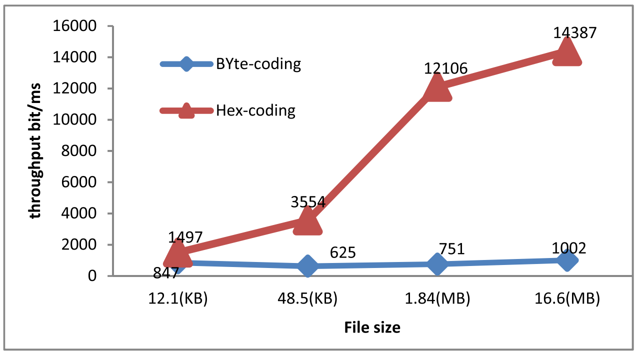
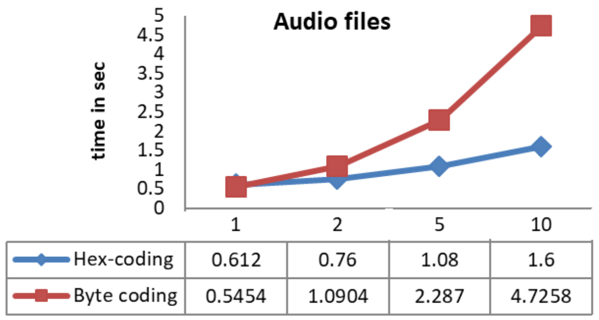


FIGURE 7. A graph comparing the throughput performance between Byte coding and hexadecimal coding method.



	1	2	5	10
Hex-coding	0.612	0.76	1.08	1.6
Byte coding	0.5454	1.0904	2.287	4.7258

FIGURE 8. A line graph illustrating the performance comparison of audio file transfer rates between Byte-coding and Hexadecimal coding methods.

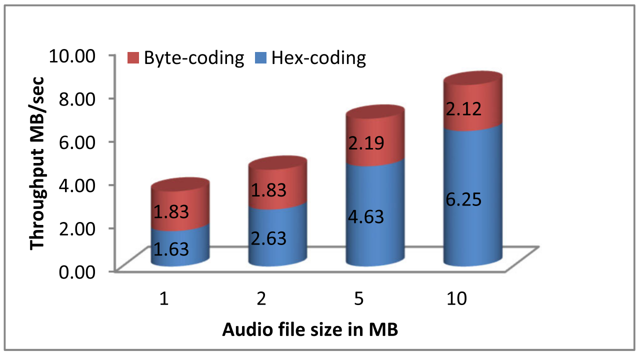


FIGURE 9. A bar chart depicting the performance comparison of audio file transfer rates between Byte coding and hexadecimal coding methods.

Figure 13 demonstrates that, when using the video format, hexadecimal optimization exhibits superior time performance compared to base64 optimization during file transfers.

Figure 14 demonstrates that combining Huffman coding with hexadecimal coding yields better performance compared to using only hexadecimal coding. The file transfer speed is significantly improved when both techniques are combined. For transferring a 12.4MB file, the hex-coding technique takes approximately 37273 seconds, while the combined technique completes the transfer in approximately 6241.5 seconds.

Figure 15 demonstrates that when hexadecimal and Huffman coding are combined, their throughput is significantly

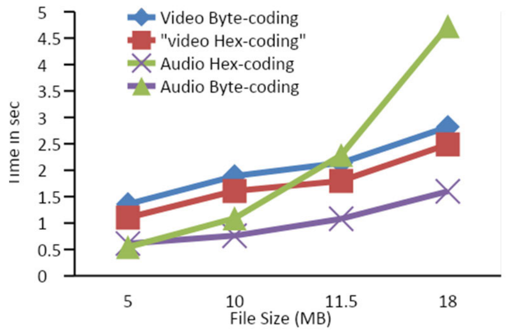


FIGURE 10. A line graph illustrating the performance comparison of audio and video file transfer rates between Byte coding and Hex coding methods.

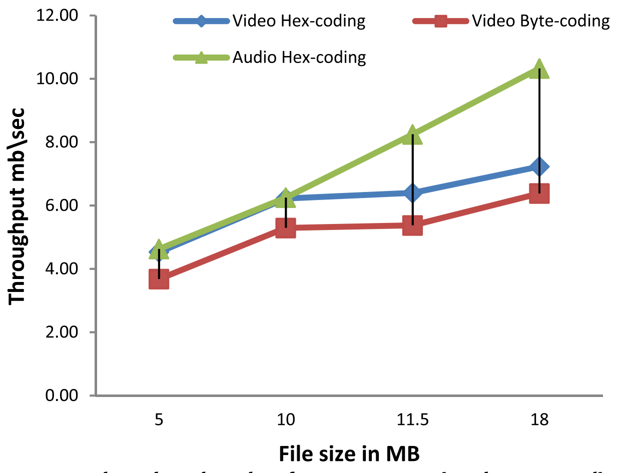


FIGURE 11. Throughput-based performance comparison between audio and video files using hexadecimal and byte coding.

higher compared to other optimization strategies utilized in this study.

C. RESULTS AND DISCUSSION

To a covert channel, we evaluate the performance of different coding techniques for transferring data in terms of time and throughput. We test various file formats, including text, audio, and video. The results are presented in Table 4.

Time measurements are recorded in milliseconds (ms), throughput is measured in bits per second, and file size is indicated in megabytes (MB). In Table 4, we present specific instances of the many experiments we conducted, covering various file formats. The tables include information on T (time in milliseconds), TP (throughput in MB per millisecond), and S (file size).

When using the text format, the convert channel delivers 12.5 MB in 104637 milliseconds, achieving a throughput of 754 bits per millisecond when employing byte coding. However, when utilizing hexadecimal coding, the time decreases significantly to 37273, resulting in a higher throughput of 6245 bits per millisecond. The best performance is achieved by combining Huffman and hexadecimal coding, where the channel delivers 12.5 MB in just 6241 milliseconds, with a throughput of 23116 bits per millisecond.

TABLE 4. Experimental results summary.

Coding techniques							
Byte		Hexadecimal		Huffman + Hexadecimal		Base64+ Hexadecimal	
T	TP	T	TP	T	TP	T	TP
12.5	104637	754	37273	6245	6241	23116	-
10	4725	20874	1600	27862		-	
10	1889	44407	1607	52200	-		189044384

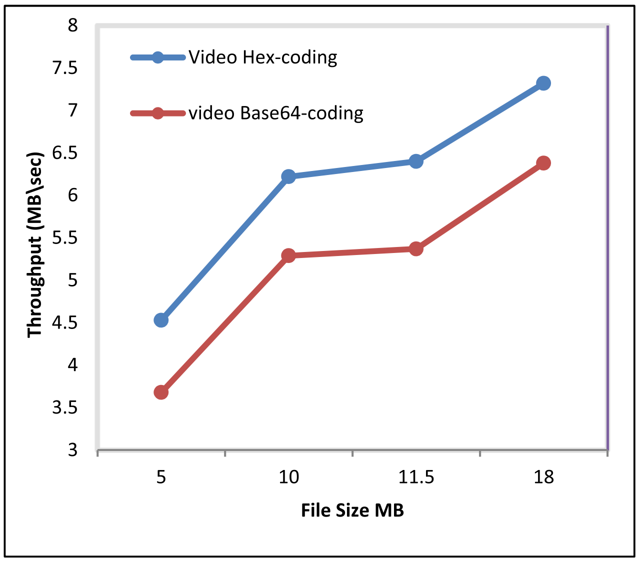


FIGURE 12. A line graph depicting the performance comparison of video file transfer throughput using Base64 coding and Hex-coding methods.

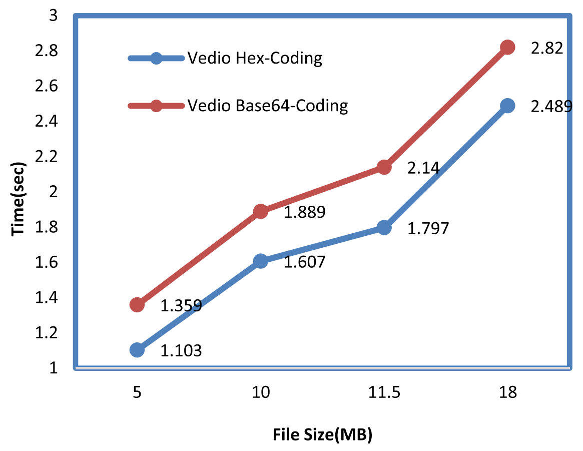


FIGURE 13. A line graph illustrating the performance comparison of video file transfer rates using Base64 coding and Hex-coding methods.

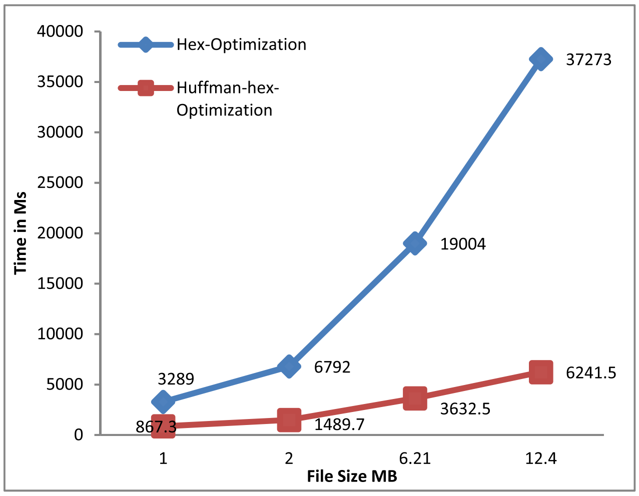


FIGURE 14. A line graph depicting the performance comparison of text file transfer rates using Huffman coding and hexadecimal coding.

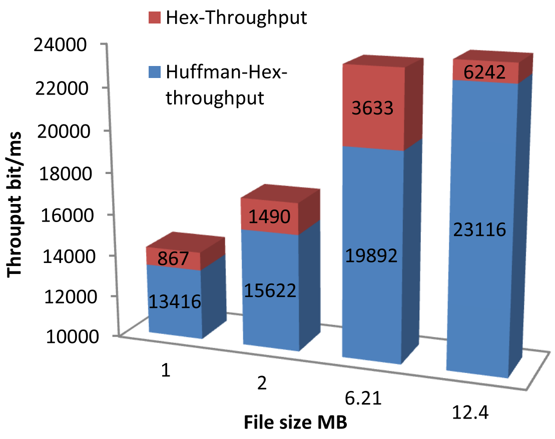


FIGURE 15. A bar chart comparing the performance of text file transfer throughput using Huffman coding and hexadecimal coding.

For audio files, the channel delivers 10 MB in 4725 milliseconds when using byte coding, resulting in a throughput of 20874 bits per millisecond. However, when employing hexadecimal coding, the delivery time decreases to 1600 milliseconds, and the throughput increases to 27862 bits per millisecond.

When it comes to video files, our covert channel delivers 10 MB in 1889 milliseconds with a throughput of 44407 bits per millisecond when using byte coding. Alternatively, when using hexadecimal coding, the delivery time further reduces to 1607 milliseconds, with a higher throughput of 52200 bits per millisecond. Lastly, when employing Base64 coding, the channel delivers 10 MB in 1890 milliseconds, achieving a throughput of 44384 bits per millisecond.

Based on the results, hexadecimal coding proves to be suitable for our covert channel due to its ability to reduce the number of attempts before encountering an Array Index Out of Bounds Exception. On the other hand, Base64 is deemed unsuitable for our covert channel as it generates larger resulting strings.

V. CONCLUSION

A covert communication channel facilitates indirect data transfer between two parties through pre-established knowledge agreements. This channel is utilized for encoding and decoding the original data and can serve both legitimate and illegitimate purposes. Many previous attempts at creating covert channel techniques have exhibited low performance in terms of throughput and time required for file delivery. To enhance data transfer performance, an appropriate encoding technique is necessary. In our proposed covert channel, we leverage exception handling in Java programs to establish and develop covert channels. We exploit the server side, which monitors the exploitation process and keeps track of the number of attempts made before encountering an ''Index Out of Bounds'' exception. Our methodology involves using several file formats, such as text, audio, and video, and implementing optimization methods, including base64, byte, hexadecimal, and Huffman coding. We evaluate the effectiveness of our covert channels by measuring the throughput and time required for file delivery. To improve performance in terms of time and throughput, we integrate optimization techniques. The sender transmits files to the server using optimization techniques, while we monitor the throughput and time required by the backdoor to receive the files. Each experiment is repeated five times, and the results are averaged. Our covert channel employs multiple optimization strategies, including byte, hexadecimal, Huffman, and base64. The results of our covert channel for data transfer using various coding algorithms demonstrate its effectiveness with various file formats, including text, audio, and video. When using the text format, our covert channel delivers 12.5 MB in 104637 milliseconds, achieving a throughput of 754 bits per millisecond when utilizing byte coding. However, the combination of Huffman and hexadecimal coding yields the best performance, delivering the same file in 6241 milliseconds with a throughput of 23116 bits per millisecond. When employing hexadecimal coding, our covert channel delivers 10 MB in 1607 milliseconds, achieving a throughput of 52200 bits/millisecond. On the other hand, when using Base64 coding, the delivery time increases to 1890 milliseconds and the throughput decreases to 44384 bits per millisecond. Furthermore, when hexadecimal coding is used with video files, our covert channel delivers 10 MB in 1607 milliseconds, with a throughput of 52200 bits per millisecond. The success of hexadecimal coding in our covert channel can be attributed to its ability to reduce the number of attempts before encountering an ''Array Index Out of Bounds Exception''. Conversely, Base64 coding is not suitable for our covert channel due to the larger resulting strings it generates.

REFERENCES

[1] S. B. Lipner, ''A comment on the confinement problem,'' *ACM SIGOPS Operating Syst. Rev.*, vol. 9, no. 5, pp. 192–196, Nov. 1975.

[2] T. S. Fatayer, S. Khattab, and F. A. Omara, ''A key-agreement protocol based on the stack-overflow software vulnerability,'' in *Proc. IEEE Symp. Comput. Commun.*, Jun. 2010, pp. 411–416.

[3] T. S. Fatayer, S. Khattab, and F. A. Omara, ''OverCovert: Using stack-overflow software vulnerability to create a covert channel,'' in *Proc. 4th IFIP Int. Conf. New Technol., Mobility Secur.*, Feb. 2011, pp. 1–5.

[4] D. J. Dye, ''Bandwidth and detection of packet length covert channels,'' Ph.D. thesis, Naval Postgraduate School, Monterey, CA, USA, 2011.

[5] T. Schmidbauer, J. Keller, and S. Wendzel, ''Challenging channels: Encrypted covert channels within challenge-response authentication,'' in *Proc. 17th Int. Conf. Availability, Rel. Secur.*, Aug. 2022, pp. 1–10.

[6] T. S. Fatayer, ''Secure communication using cryptography and covert channel,'' in *Computer and Network Security*. IntechOpen, 2020.

[7] E. S. Lee, ''Essays about computer security,'' Tech. Rep., 1999, p. 181.

[8] J. Wu, Y. Wang, L. Ding, and X. Liao, ''Improving performance of network covert timing channel through Huffman coding,'' *Math. Comput. Model.*, vol. 55, nos. 1–2, pp. 69–79, Jan. 2012.

[9] S. G. Josefsson, *The Base16, Base32, and Base64 Data Encodings*, document RFC 4648, 2006, pp. 1–44.

[10] A. Alhelal and M. Al-Khatib, ''Systematic analysis on the effectiveness of covert channel data transmission,'' *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 5, 2023.

[11] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, ''Pattern-based survey and categorization of network covert channel techniques,'' *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–26, Apr. 2015.

[12] S. Zander, G. Armitage, and P. Branch, ''A survey of covert channels and countermeasures in computer network protocols,'' *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 44–57, 3rd Quart., 2007.

[13] J. Saenger, W. Mazurczyk, J. Keller, and L. Caviglione, ''VoIP network covert channels to enhance privacy and information sharing,'' *Future Gener. Comput. Syst.*, vol. 111, pp. 96–106, Oct. 2020.

[14] M. Akil, L. V. Mancini, and D. Venturi, ''Multi-covert channel attack in the cloud,'' in *Proc. 6th Int. Conf. Softw. Defined Syst. (SDS)*, Jun. 2019, pp. 160–165.

ABDULRAHMAN ALHELAL received the bachelor's degree in computer science from Imam Mohammad Ibn Saud Islamic University, where he is currently pursuing the master's degree with the Faculty of Computer and Information Sciences. He has a publication in the security field. His research interests include information security, cryptography, coding, steganography, and covert channels.

MOHAMMAD AL-KHATIB received the bachelor's degree in computer science from Irbid National University, the master's degree in information systems from DePaul University, and the Ph.D. degree in computer science, specializing in security in computing from University Putra Malaysia (UPM). He is currently an Assistant Professor with the Faculty of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University. His research interests include information security, cryptography, the Elliptic Curve algorithm, and information retrieval.