CPSOCKS: Cross-Platform Privacy Overlay Adapter Based on SOCKSv5 Protocol

Gabriel-Cosmin Apostol[1], Alexandra-Elena Mocanu[1], Bogdan-Costel Mocanu[1], Dragos-Mihai Radulescu[1], and Florin Pop[1,2(✉)]

[1] Department of Computer Science, University Politehnica of Bucharest, Bucharest, Romania
{gabriel.apostol,dragos.radulescu}@stud.acs.upb.ro,
alexandra_elena.mihaita@cti.pub.ro,
{bogdan_costel.mocanu,florin.pop}@upb.ro

[2] National Institute for Research & Development in Informatics - ICI Bucharest, Bucharest, Romania
florin.pop@ici.ro

Abstract. In figures, the cybersecurity landscape is one of the most across-the-border impactable trends in the last years, especially after the begging of the COVID-19 pandemic. Therefore, by the end of Q4 of 2021, more than 281 million people have been victims of data breaches and cyber-threads, costing more than $42.96 million per day. A possible explanation is that most network operators do not provide any mechanism that blocks path tracing. Almost anybody with above-average network security knowledge can use public path tracing tools such as *traceroute*, enabling malicious users and thread factors to craft sophisticated cyber-attacks easily. Therefore, this paper proposes a cross-platform privacy overlay over the SOCKSv5 protocol. We evaluate the proposed solution in terms of latency, average throughput, and transfer rate.

Keywords: CSOCKS5 · Cloud network access patterns · Obfuscation · Privacy · Security

1 Introduction

Obfuscation of data in network transmissions represents a major research domain since it covers a large span of techniques meant to make sensitive data harder to detect. Due to internet censorship in some states, data-hiding techniques have evolved, making them less susceptible to detection. However, it is a well-known fact that there is a continuous innovation effort ongoing, both for censors and privacy advocates. One important aspect of network obfuscation is the fact that is a critical process that needs to be executed in time constraint fashion.

The information hiding domain has been researched since ancient times, and various methods have been devised since then, like the invisible ink, or the encoding of letters in music notes. More elaborate methods have been devised during

C. Yu et al. (Eds.): GPC 2022, LNCS 13744, pp. 149–161, 2023.
https://doi.org/10.1007/978-3-031-26118-3_11

the two world wars, since the army research programs have come up with many innovations, like code-words and microdots, which were microscopic texts hidden in punctuation marks. In the digital era, the increasing interest in privacy has led to a relatively large number of information-hiding methods targeting both network transmission and files.

The purpose of the research in the domain of network access patterns obfuscation is to identify novel methods meant to bypass censorship and surveillance. Since full online anonymity is not easily achievable and assumes a user must change some habits, it is important to think about people who want to achieve it. In many cases, it has been proven that they are individuals breaking laws, and these facts have led to ethical debates regarding the types of concealed activities conducted using existing technologies. On the other hand, many oppressive regimes have been exposed by whistle-blowers, which relied on traffic obfuscation technology to deliver real information worldwide.

Another positive outlook of online privacy-enhancing technologies is given by the fact that sensitive aspects of the personal life of a user can be concealed from internet service providers. The most relevant example is the access to personal health-related data, which must not be necessarily secret, but shared only between the involved parties.

It has been widely debated that internet censorship will be employed in various countries in the foreseeable future. Researchers and industry experts should continue working to craft a future in which privacy prevails against nationwide censors, providing people with non-discriminatory access to information. One of the purposes of network traffic obfuscation employment is to help circumvent abusive surveillance and censorship. Sometimes, these techniques are used even by Law Enforcement to perform sting operations without disclosing IP ranges associated with the police, as mentioned in [2].

The biggest challenge that researchers face is the lack of an adversary model. This fact hinders efforts to provide a reliable and resilient censorship bypass system and prevents researchers from realizing if a new obfuscation method is vulnerable in practice. Another research issue is the design of such systems since tradeoffs among security, performance, and ease of use must be made.

To understand restrictions imposed by the censors, researchers should have access to deep packet inspection systems employed in various countries, an impossible fact. This drawback is a speed diminishing factor for research, but at the same time, might be considered a source for various novel techniques, useful for privacy enhancement.

Another approach to developing better obfuscation methods is to understand what normal traffic means for a censor. To achieve this, scholars should be allowed to analyze large amounts of traffic captured from retention systems, but this would have an enormous impact on user data privacy.

However, a series of questions arise regarding user interaction with network obfuscation tools. It is not yet clear how users are perceiving such tools from the perspective of risks, ethics, and legal constraints. Moreover, it is not clear how people will have access to this type of software. Since the regular users

could be censors as well, it is not clear how much time would take to identify, detect and defeat a new layer of obfuscation. Another open research problem is providing users with means of plausible deniability, in case they must support legal consequences in oppressing countries.

The rest of the paper is structured as follows. In the second Section, we present a detailed state-of-the-art analysis, while in the third Section, we briefly describe the SOCKSv5 protocol, on which our solution is based. Furthermore, in forth Section we present our cross-platform obfuscation solution and in the fifth Section we evaluate it. Finally, in the sixth Section, we conclude our research.

2 Related Work

To hide data in legitimate network transfers, researchers have analyzed various techniques, targeting both the timing and storage of the protocols. Any shared resource in a network can be eligible for designing a covert channel, meant to evade deep packet inspection.

Covert channels are deaplly covered as a part of the discipline called steganography, and in network environments, can be classified according to multiple criteria, depending on the way, they transfer covert data, by their reliability, by the network layer they occur or by their ability to generate additional traffic or modify the existing one.

Timing channels are created by inducing specific events at a given time and require both the receiver and the sender have internal reliable time sources. On the other hand, a storage channel is created by modifying a shared resource, directly or indirectly.

A covert timing channel involves the evaluation of specific events occurring in various protocols [9] at a certain moment. They can be achieved by manipulating the delivery times of various network layer messages, though not very reliable if the delivery route changes dynamically (load balancing, dynamic routing protocols), thus inducing unwanted noise.

In [2], the authors propose a novel real-time solution for hiding the cloud network access patterns using a chatbot implemented on the SOCKS5 protocol. An interesting use case of using chatbots in emergency situations is also presented in [15].

A covert storage channel can be achieved by modifying the fields of various protocols [10], in a manner that would not impact the transmitted data. Such examples include tunnels, which are essentially a manner of encapsulating a covert protocol in the payload data of other protocols, like HTTP or HTTPS.

Data hiding could occur in network communications starting at the network layer in the TCP/IP stack, starting with the fundamental protocols in the IP suite, including ICMP, TCP, UDP, and DNS.

ICMP (Internet Control Message Protocol) has been proposed as a good candidate, and the most common software using this technique is called Loki [4]. This software can generate ICMP packets and control their payload (Fig. 1), to encapsulate other protocols. However, this covert channel can be easily detected

by employing DPI (deep packet inspection), by applying firewall rules, or by employing an IDS (intrusion detection system). Also, a traffic analysis captured on a larger period would yield a spike in the number of ICMP requests.

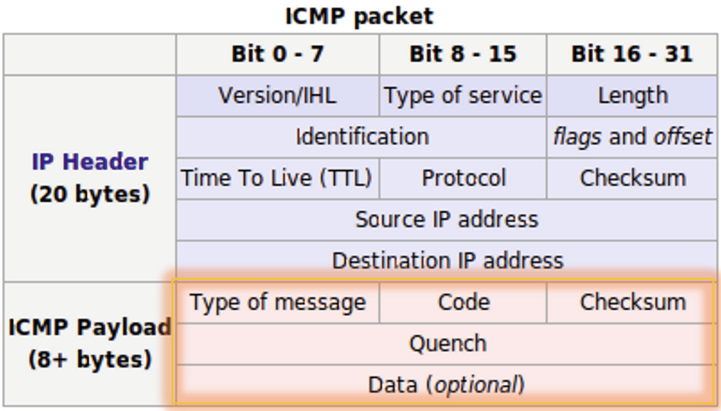

Fig. 1. ICMP packet structure.

DNS (domain name system) tunneling has been around for more than a decade [8] and can be conducted by manipulating the name of the desired domain in the payload of the request message (). This technique can be detected by DPI systems [20], and the analysis of a capture file could provide valuable insight because the domain names are encoded using the base32 scheme.

IP steganography [11,19] assumes the manipulation of the IP header, to deliver data covertly. The eligible field proposed for this technique is called Flags whitch is used to signal a value of zero or one at a certain offset, called DF (Do not fragment). This method has a drawback since the sender and receiver must know the size of the MTU (maximum transport unit) in advance.

The TCP/IPv4 protocol could be misused [1,3] to ensure a reliable covert channel, by altering the padding bits (PAD), the urgent pointer, the reserved bits, or even by using port numbers as an alphabet (Fig. 2).

Another approach in the field of network communication steganography is the repurposing of existing layer four protocols. Therefore, many software implementations allow the cloaking of various protocols using HTTP, HTTPS, SSH, FTP, RTP, RTCP, SIP, and UDP. HTTP and HTTPS can be easily manipulated by the initiator by altering both headers and payloads. There are many methods of manipulation, starting from low-noise techniques, like changing the order of the headers to entirely modifying the request body or parameters. The detection methods rely on statistical traffic attributes [14], since these protocols are designed for great flexibility. SSH tunneling is aimed to provide two layers of protection, by employing encryption and acting as a proxy, without revealing the actual destination endpoints. Usually, SSH outbound connections are denied

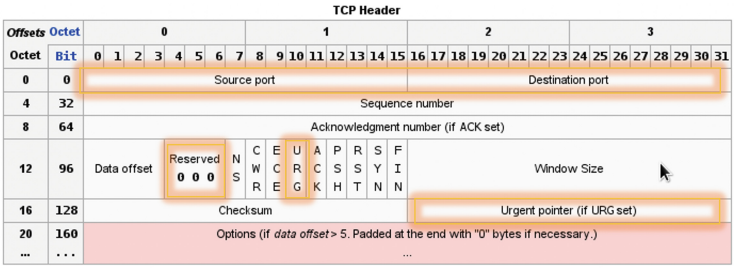

Fig. 2. TCP header structure.

due to these factors. The detection of suspicious traffic has been implemented using GMM (Gaussian Mixture Model), which, under reasonable assumptions, can be used to predict which application is being tunneled [7].

Another interesting approach ca be the usage of Peer-to-Peer tools like TOR [6], I2P [21], FreeNet [5], and UPB developed overlays like: SPIDER [16], HoneyComb [18] and AFT [17].

3 SOCKS v5 Protocol

SOCKSv5 protocol, defined in the RFC-1928 [13], relays on the compilation and the linking of a client application to the appropriate encapsulation routines of the SOCKS library. Since the latest version of this protocol, both TCP and UDP-based applications are supported. Also, this version provides strong authentication methods and addressing schemes to include IPv6 address and the domain name.

When a TCP-based client initiates a connection to a resource that is reachable only via a firewall, the application opens a TCP connection to the appropriate port on the SOCKS server system. By default, the SOCKS service is accessible on port 1080, as mentioned in [12]. If the connection request is successful, the client starts a negotiation process for the authentication method which will be further used, and then sends a relay request. After that, the SOCKS server evaluates the request, and accept or deny the connection. When the TCP-based client application connects to the SOCKS server, it sends a message which contains the following parameters, as presented in [12]:

```
VER : 1 byte;
NMETHODS : 1 byte;
METHODS : 1 - 255 bytes.
```

The VER value is set with the X'05' value to indicate the latest version of the SOCKS protocol. The NMETHODS field specifies the number of identifiers contained in the METHODS field.

The values defined, in the RFC file [12], for the METHOD field are:

– X'00' : NO AUTHENTICATION REQUIRED;
– X'01' : GSSAPI;
– X'02' : AUTHENTICATION SCHEME using username and password;
– X'03' - X'7F' : IANA ASSIGNED;
– X'80' - X'FE' : RESERVED FOR PRIVATE METHODS;
– X'FF' : NO ACCEPTABLE METHODS.

The SOCKS server selects one of the given methods specified in the METHODS field and then sends a method selection message, with the following scheme:

```
VER : 1 byte;
METHODS : 1 byte.
```

In the METHOD is X'FF', it means that none of the methods selected by the client are acceptable and therefore the client closes the TCP connection. Once the method negotiation process has been completed, the client sends the request details. If this phase, if the agreed method includes confidentiality and integrity validation, these requests are performed using encapsulation-dependent methods.

The request packet is formatted as follows:

```
VER : 1 byte;
CMD : 1 byte;
RSV : 1 byte;
ATYP : 1 byte;
DST.ADDR : variable;
DST.PORT : 2 bytes.
```

where:

– VER : Protocol version (X'05');
– CMD : The selected command, which can be one of the following values:
  • CONNECT : X'01';
  • BIND : X'02';
  • UDP ASSOCIATE : X'03'.
– RSV : Reserved field : X'00';
– ATYP : Address type:
  • IPv4 address : X'01';
  • Domain name address : X'03';
  • IPv6 address : X'04'.
– DST.ADDR : Destination IP address or domain name;
– DST.PORT : Destination port;

Moreover, the SOCKS server evaluates the request using the source and destination addresses and answares with one or more messages. The message transmitted by the server is formed as follows:

```
VER : 1 byte;
REP : 1 byte;
RSV : 1 byte;
ATYP : 1 byte;
BND.ADDR : variable;
BND.PORT : 2 bytes.
```

where

- VER - Protocol version : X'05';
- The reply field, which can be one of the following values:
  - Success : X'00';
  - General SOCKS server failure : X'01';
  - Connection not permitted : X'02';
  - Network unreachable : X'03';
  - Host unreachable : X'04';
  - Connection refused : X'05';
  - TTL expired : X'06';
  - Command not supported : X'07';
  - Address not supported : X'08';
  - Unassigned value : X'09' - X'FF'.
- RSV : Reserved field : X'00';
- ATYP : Address type:
  - IPv4 address : X'01';
  - Domain name : X'03';
  - IPv6 address : X'04'.
- BND.ADDR : Bound server address;
- BND.PORT : Bound server port.

If the negotiated method in the authentication process includes integrity and confidentiality validations, the replies will use the encapsulation-dependent methods.

In the address field (BND.ADDR, DST.ADDR), the ATYP value specifies the type of address:

- X'01' : an IPv4 address (32 bits);
- X'03' : a fully-qualified domain name. The first byte contains the size of the name, followed by the value itself;
- X'04' : the address is an IP v6 address, with a length of 16 octets.

In reply to a CONNECT command sent by a client, the BND. The PORT field contains the port number for the SOCKS server and the BND.ADDR field contains its associated IP address. Usually, the BND.ADDR value is different from the IP address used, by the client, to reach the server, since these servers are often located in multiple places.

4 Proposed Solution

Our implementation aims to provide easy integration with existing applications, by combining a local SOCKS5 proxy with various legitimate protocols which allow input manipulations. Such protocols are used mostly by text chat services. These protocols will further carry data toward the final endpoint, mediated by various service providers. The endpoint shall be able to listen to incoming messages and decode them accordingly. After the endpoint decodes the requests, it will perform connections in the name of the client and deliver the data through the same protocols used by the request.

4.1 Use Case

We developed a SOCKSv5 proxy to route existing applications' network requests through a privacy overlay designed to hide network access patterns. To describe the interactions between the actors and the system, we represented a part of the functionality by including a use case diagram.

The use case starts when the actor opens the local privacy overlay application as shown in Fig. 3. Firstly, he must choose from the user interface one of the available IPv4 addresses, which will be used to listen for SOCKv5 requests.

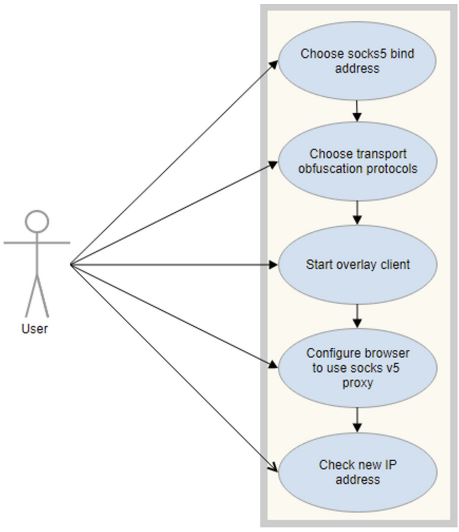

Fig. 3. CPSOCKS use case diagram.

Next, the actor must choose the transport obfuscation protocols. This method is not implemented yet, so the server simply acts like a regular SOCKSv5 proxy server.

After the bind address has been selected, the user can start the local SOCKSv5 server, allowing proxy-aware applications to use it.

To make use of our software implementation, the actor must also configure the desired applications with the address and port of the proxy server.

The last step is the verification process when the user can check the exit IP address if the application allows it. This statement is specifically true when it comes to web browsers since there are websites that allow the user to view his IP address.

4.2 Software Architecture

In the current development iteration, we managed to implement and benchmark a cross-platform SOCKS5 proxy server as shown in Fig. 4, which will be used to capture application-generated data. Our proxy server uses a limited set of SOCKS5 instructions. Currently, it supports only the TCP CONNECT method and provides no authentication method.

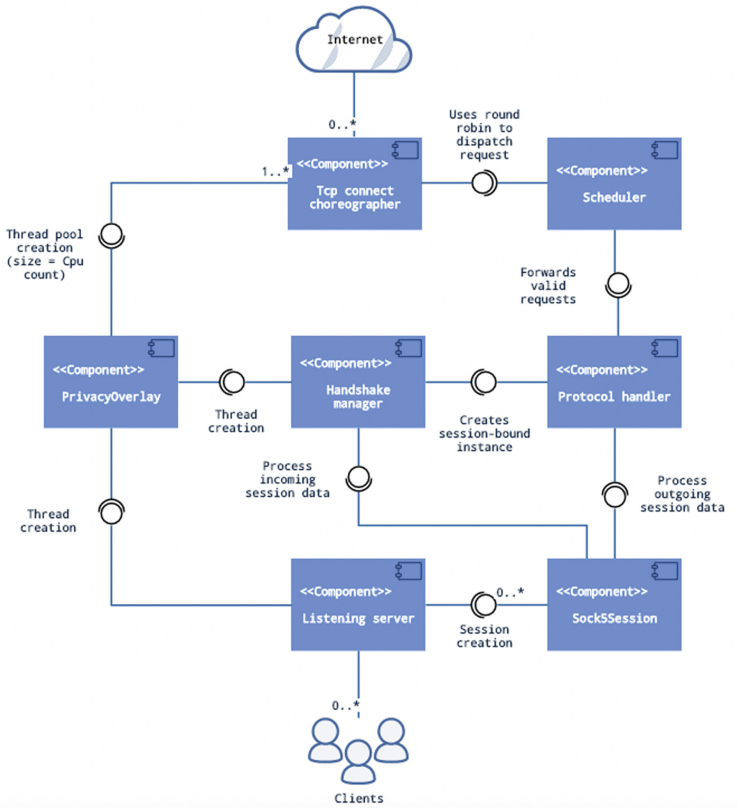

Fig. 4. CPSOCKS software architecture.

The language we chose for our implementation was Java, using the Java Development Kit version 17, mainly due to its cross-platform nature. We decided to use Java.NIO package to implement a scalable server, with a fixed number of threads and non-blocking input/output operations.

The listening server is running in its own dedicated thread, accepting connections on a specific port, enqueuing the incoming data, and dequeuing outgoing data using session-bound queues. The handshake manager runs in a separate thread, polling session queues. If any data has been pushed by the server, it employs a session-specific protocol handler to evaluate the requests and writes back its resolution afterward.

To improve speed, we decided to use multiple TCP connect choreographers, each one running in its own thread. When a connect request is dispatched, a round-robin scheduling strategy is employed to equally distribute the connection requests. The number of threads is variable, depending on the CPU cores count.

A TCP choreographer is responsible for the creation and management of multiple connections. It also routes the data generated by the actual requests and the listening server.

5 Use Case and Experimental Evaluation

Our software implementation has been load-tested by simulating a high number of concurrent requests. Since a single instance of a bench-marking solution could not provide a sufficient load to prove the performance of our implementation, we decide to employ distributed testing (Fig. 5 Testing architecture).

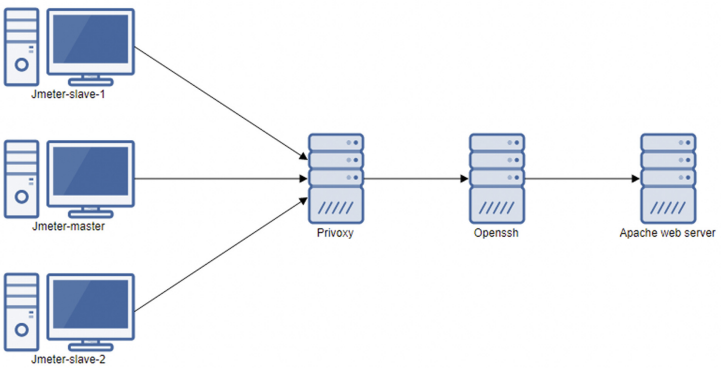

Fig. 5. CPSOCKS testing architecture.

We identified multiple load-testing solutions, but we decided to use Apache JMeter, mainly because it can orchestrate multiple bench-marking agents. However, the only downside this software implementation has is given by the fact

that it supports only HTTP proxies. As a workaround, we decided to use a translation layer capable to convert HTTP proxy requests into SOCKS5 requests by using Privoxy, a software tool allowing such operations.

The benchmark we conducted had the purpose to compare our software implementation with OpenSSH, a multi-purpose application that can act as a SOCKSv5 proxy server.

The testing scenario has yielded an overall lower performance, in terms of latency, throughput, and transfer rate as shown in Fig. 6. This is happening because we are handling data transfer with additional round trips from kernel to user space. This design is acceptable because the data shall be further transformed before delivery through alternative transport channels. Also, the proxy will be accessed by a single user having a limited set of applications.

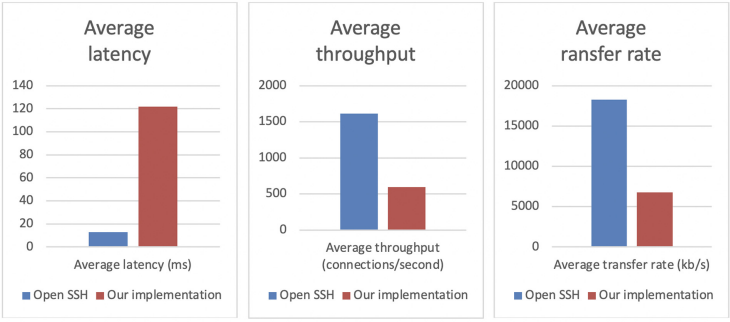

Fig. 6. CPSOCKS benchmark results.

6 Conclusions

Overlay networking, also known as software-defined networking, is a vastly-employed network-enhancing technique used to define new protocols over existing ones. The main advantage of overlay networks is that they can be defined using software, opening new opportunities in the field of security research and advancements. The most valuable enhancements they can offer are customizable encapsulations and routing protocols.

Our solution aims to provide a platform-independent method of integration with software-defined networks by leveraging the existing SOCKS-5 protocol. Even if the idea is not new, we needed a software solution that had no dependencies on external libraries and a relatively small, maintainable, and easy-to-understand code base.

The components created in this iteration are reusable and will be included in future software releases. However, they will be subject to further enhancements, mainly because now they are processing data at byte-level, rather than block-level. To achieve that, the input and output queues shall be re-engineered performance-wise.

Even if the current performance is lower than we expected, we consider that the proxy server is performing reasonably well for a single user which is not using data-intensive applications. Even though our experimental results show a reduction in performance, we still consider that our approach is an optimal solution for real-time scenarios.

Acknowledgements. The research presented in this paper was supported by project FARGO: Federated leARninG for human moBility (PN-III-P4-ID-PCE-2020-2204), the project CloudPrecis (SMIS code 2014+ 124812), project ODIN112 (PN-III-P2-2.1-SOL-2021-2-0223) and by the University Politehnica of Bucharest through the Pub-Art program. Also, this paper was partially supported by the grant POCU/993/6/13-153178, co-financed by the European Social Fund within the Sectorial Operational Program Human Capital 2014-2020, and the DECIP project (contract no. PFE57/2022).

We would also like to thank the reviewers for their valuable comments and insights.

References

1. Ahsan, K., Kundur, D.: Practical data hiding in TCP/IP. In: Proceedings of the Workshop on Multimedia Security at ACM Multimedia, vol. 2, no. 7, pp. 1–8. ACM Press, New York (2002)
2. Apostol, G.-C.: Hiding cloud network access patterns for enhanced privacy. In: 2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet), pp. 1–5. IEEE (2022)
3. Arthur, M.: Steganology and information hiding: stegop2py: embedding data in TCP and IP headers (2021)
4. Berghel, H.: Hiding data, forensics, and anti-forensics. In: Communications of the ACM, vol. 50, no. 4, pp. 15–20 (2007)
5. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: a distributed anonymous information storage and retrieval system. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44702-4_4
6. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second- generation onion router. Technical report, Naval Research Lab Washington DC (2004)
7. Dusi, M., Gringoli, F., Salgarelli, L.: A preliminary look at the privacy of SSH tunnels. In: 2008 Proceedings of 17th International Conference on Computer Communications and Networks, pp. 1–7. IEEE (2008)
8. Dusi, M., et al.: Detection of encrypted tunnels across network boundaries. In: 2008 IEEE International Conference on Communications, pp. 1738–1744. IEEE (2008)
9. Gray III, J.W.: Countermeasures and tradeoffs for a class of covert timing channels. Technical report (1994)
10. Kemmerer, R.A., Porras, P.A.: Covert flow trees: a visual approach to analyzing covert storage channels. In: IEEE Transactions on Software Engineering, vol. 17, no. 11, p. 1166 (1991)
11. Kundur, D., Ahsan, K.: Practical Internet steganography: data hiding in IP. In: Proceedings of the Texas WKSP. Security of Information Systems (2003)
12. Leech, M., et al.: RFC1928: SOCKS Protocol Version 5. (1996)
13. Leech, M., et al.: SOCKS protocol version 5. Technical report (1996)

14. Lin, H., Liu, G., Yan, Z.: Detection of application-layer tunnels with rules and machine learning. In: Wang, G., Feng, J., Bhuiyan, M.Z.A., Lu, R. (eds.) SpaCCS 2019. LNCS, vol. 11611, pp. 441–455. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24907-6_33
15. Mocanu, B.C., et al.: ODIN IVR-interactive solution for emergency calls handling. Appl. Sci. **12**(21), 10844 (2022)
16. Mocanu, B., et al.: SPIDER: a bio-inspired structured peer-to-peer overlay for data dissemination. In: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 291–295. IEEE (2015)
17. Poenaru, A., Istrate, R., Pop, F.: AFT: adaptive and fault tolerant peer-to-peer overlay—A user-centric solution for data sharing. Futur. Gener. Comput. Syst. **80**, 583–595 (2018)
18. Pop, F., et al.: Trust models for efficient communication in mobile cloud computing and their applications to e-commerce. Enterp. Inf. Syst. **10**(9), 982–1000 (2016)
19. Sengupta, A., Rathor, M.: IP core steganography for protecting DSP kernels used in CE systems. IEEE Trans. Consum. Electron. **65**(4), 506–515 (2019)
20. Zander, S., Armitage, G., Branch, P.: A survey of covert channels and countermeasures in computer network protocols. IEEE Commun. Surv. Tutorials **9**(3), 44–57 (2007)
21. Zantout, B., Haraty, R., et al.: I2P data communication system. In: Proceedings of the ICN. Citeseer, pp. 401–409 (2011)