Data Exfiltration: Methods and Detection Countermeasures

James King*, Gueltoum Bendiab*, Nick Savage*, Stavros Shiaeles*
*Cyber Security Research Group, University of Portsmouth, PO1 2UP, Portsmouth, UK
james.king2@myport.ac.uk, gueltoum.bendiab@port.ac.uk,
nick.savage@port.ac.uk, sshiaeles@ieee.org

***Abstract*—Data exfiltration is of increasing concern throughout the world. The number of incidents and capabilities of data exfiltration attacks are growing at an unprecedented rate. However, such attack vectors have not been deeply explored in the literature. This paper aims to address this gap by implementing a data exfiltration methodology, detailing some data exfiltration methods. Groups of exfiltration methods are incorporated into a program that can act as a testbed for owners of any network that stores sensitive data. The implemented methods are tested against the well-known network intrusion detection system Snort, where all of them have been successfully evaded detection by its community rule sets. Thus, in this paper, we have developed new countermeasures to prevent and detect data exfiltration attempts using these methods.**

***Index Terms*—Data exfiltration, security, Cyberattack, steganography, countermeasures, Intrusion Detection System.**

I. Introduction

Currently, data exfiltration, also referred to as data theft or data leakage, is a serious threat to individuals, companies and organisations, as the number of incidents continues to increase dramatically worldwide [1], [2]. This security threat refers to data breaches that occur when a malicious actor performs an unauthorised transfer or retrieval of private information (e.g. intellectual property, trade secrets, financial records and customer information) from a computer/device or a server [3], [4]. This can be done manually, by an individual with physical access to the system, or remotely through malicious software over the network [1], [3]. Data exfiltration has become a challenging security issue as new evolutions of well-crafted cyber-attacks are designed to remain undetected for a long period of time, with potentially destructive consequences [1]. A recent report by TESSIAN announced 47% increase in incidents over the last two years; this includes accidental data loss and deliberate data exfiltration by negligent or dissatisfied employees and bad actors [5]. In addition, the increasing growth in smart mobile and IoT devices usage has lead to a higher possibility of data breaches, as over 98% of these devices are vulnerable and can be easily exploited in data exfiltration attacks [6].

A wide variety of technical methods have been already used by attackers to steal data from individual devices and organisation's systems [1]. However, only a small number of existing studies and security reports focused on demonstrating data exfiltration operations [7], [8]. For instance, authors in [9] demonstrated that DNS queries can be used to perform data exfiltration out of a network. In [10], authors presented three types of covert data exfiltration methods that can be applied to industrial networks. The methods are storage-based, behaviour-based and timing-based. This paper highlighted the potential damage and easy exploration of these methods to exfiltrate data. Some other studies focus on using images and video steganographic schemes for data exfiltration [11], [12]. In general, the data exfiltration threat has not been deeply explored, which is very important for future design and implementation of security solutions.

In this paper, we will address this gap in the literature by developing a data exfiltration methodology that uses different data exfiltration methods, including image steganography, HTTP 404 errors on web logs, Time to Live (TTL) manipulation, IP options field manipulation, hiding data in email headers and timing control. The studied methods are implemented and tested against the well-known IDS Snort, where all of them have successfully evaded detection by the snort rule set. In addition, detection countermeasures have been proposed to detect and prevent such attack vectors. In this context, a new Snort rule set has been developed along with other countermeasures that will help to increase the accuracy of the Snort IDS by reducing the false-positive rate that the new rule set will likely cause when applied in an enterprise setting with large and varied network traffic. The remainder of this paper is structured as follow: In the following section, we introduce related work on data exfiltration prevention solutions and countermeasures. The third section outlines the proposed methodology to implement the data exfiltration methods studied in this paper. While, the experiment setup, testing results and proposed countermeasures are described in the fourth section. Finally, section five concludes the paper, proposes some future work and open issues.

II. Related work

Data exfiltration is a highly active research area where various security systems for preventing and detecting data theft have been proposed in the literature [13]–[15]. studies in [3], [13] examined existing Data Exfiltration Prevention Systems (DEPSs), by focusing on their challenges and countermeasures. In [3], authors proposed potential areas that could be considered in future research such as content analysis, internal misuse, and smartphone security. In the same context, authors in [14] provided an overview of the most well

known information-hiding mechanisms that can be utilised by malware. Another recent study in [2] presented a taxonomy of data exfiltration prevention solutions in both industry and academic research areas. This study highlighted the causes and motivations for data leakage, types of leaked information and data leakage channels. Authors in [1] critically analysed the existing data exfiltration attack vectors and countermeasures, intending to report the current state of the art in this area and determine gaps for future research. In order to detect data exfiltration attacks and stop them, some approaches inspect known channels that could be exploited by attackers for stealing sensitive information such as emails [16], HTTP/HTTPS downloads and uploads, DNS protocol [9], [17] and some features of the TCP/IP protocol suite [18]. For instance, authors in [19] proposed a novel DNS exfiltration detection approach based on machine learning, which targets both DNS tunnelling and low throughput malware exfiltration. The method inspection is based on primary domains, which allows filtering of legitimate services that exchange data over the DNS. The validation of this method is done on two DNS tunnelling tools and two DNS exfiltration malware injected in large-scale DNS traffic. Some other solutions have used the DPI (deep packet inspection) method to monitor network traffic and detect data exfiltration [20]. This method monitors all outgoing traffic for sensitive data and provides a higher level of detection capability as it ensures that not a single data packet goes un-inspected [1]. However, inspecting every packet going out of the network, makes it unsuitable for use in some environments, as it is resource-intensive and can lead to critical data privacy implications.

Digital steganography is one of the most exploited techniques to exfiltrated sensitive data from an internal network [21], [22]. It is usually used by hackers to encode or embed sensitive information inside image [23] or video files [24]. In this context, several methods have been proposed to prevent and detect data leakage attempts using this technique. For instance, authors in [24] proposed an approach to inspect data hidden in the video frames (i.e., data frames, metadata and audio stream) by using Discrete sine transform (DST) and wavelet filters. However, the intensive filtering for each video going out of the internal network could cause a significant delay and damage to the quality of the communication. Cloud services, especially public services, have also introduced new categories of data exfiltration risks that should be considered by security researchers. In this context, a number of methods have focused on the detection of data exfiltration by applying anomaly-based detection mechanisms to the cloud and profiling the normal behaviour of users [25], [26]. However, most of them have not been implemented in a real cloud environment.

III. Methodology

This section describes the proposed methodology for implementing the selected data exfiltration methods. In all the implemented methods, there are two different parts, sending and receiving. The sending part uses an exfiltration method for sending the files to be exfiltrated from the disk, whereas the receiving part uses the reassembling method related to the selected exfiltration method. In the proposed methodology, we can read and exfiltrate any file format from the disk in a binary format. The sending machine can also add a time delay between the sending of data when it uses a network connection in order to avoid detection. The implementation of the proposed methods is done using the Python language as it is able to communicate with the lower-level parts of the networking layers, which is required in this work. It also a has higher-level language utility that would be useful for implementing the selected exfiltration methods such as the Scapy library that will be used to craft, send, and receive packets.

A. Data exfiltration methods

There exist a large number of data exfiltration methods that have already been presented in the literature [1], [14], [15], [21]. However, in this work, we only implemented potential data exfiltration methods of both newly designed and previously described [16], but newly streamlined methods. As illustrated in Fig. 1, these methods can be divided into three main categories: content-based, header-based, and meta-based methods.

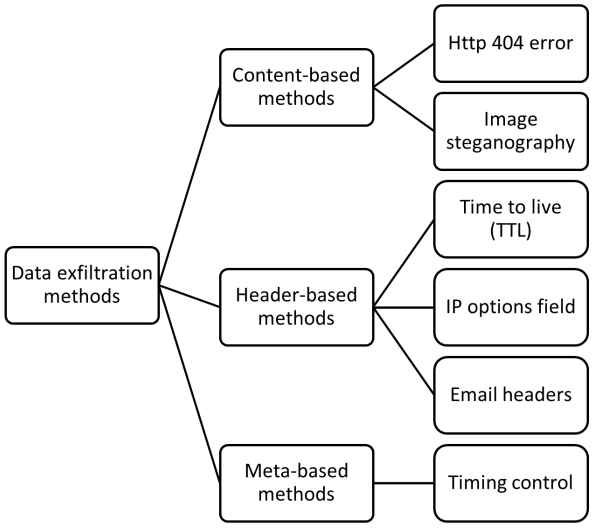

Fig. 1. Data exfiltration methods

1) Content-based methods: One of the simpler methods for data exfiltration attempts is to hide the exfiltrated information in the sent content, where it can be encoded without the content looking malformed [1]. In this paper, we studied two content-based methods: HTTP 404 error, image steganography.

Http 404 error: In this method, data can be exfiltrated using HTTP 404 errors, which are the typical response of a web server for a resource that does not exist [21]. First, the file is read from the disk in binary form. Then, each byte is converted into a two-digit hexadecimal number. After that, the generated hex numbers are combined into strings of a length that is defined by the hardcoded value of the "URLlength". This value is generally set to 500 and can be changed at the top of the

source code. Ten of these sequential hexadecimal numbers are then concatenated and formatted into a URL, which is then sent to the user-defined host address. In the implemented method, the sending part first selects the file to be extracted from the disk, the destination web address and a possible delay. Then, a simple get request is sent to check whether the receiver is reachable or not. After that, the file is read as a binary array and converted into a URL as described before. When a URL of the specified length is generated, it is appended to a new URLs output array. A part number is appended onto the front of the URL and separated from the concatenated hexadecimal values with a "/". The file name goes through a similar process where it is converted into a hexadecimal representation of ASCII and then, added to a URL with the part number of "n". This URL then gets appended to the URL output array. After creating the complete URL, it is sent to the receiver using the defined delay. The testing of this method is done by using the Amazon web services (AWS) S3 bucket [27] with public permissions to view the contents as a static web page. The S3 bucket also has server-access logging enabled, to log all connections and requests to the S3 bucket. This makes it act in a similar way to a web server and its logging conventions. In this work, the part number embedded into the sent URL is used by the receiver to reassemble the received information in the corrected order. As the logs coming from an S3 bucket, they will not be in the correct order. Further, the S3 buckets logging stores more information than needed, thus, the receiver, will extract just the requested URL into a new array.

Image steganography: Images provide complex files that can be used to hide information, where the output image is visually identical to the original one [22]. Image steganography usually hides the exfiltrated data in the least significant bit of each colour value, by moving down the columns of pixels encoding three bits to every pixel, putting one bit in the least significant bit of the values in the RGB channels. An efficient method to modify the least significant bit is to check whether the initial value is odd or even. If it is odd, the least significant bit will be set to one, otherwise, it will be set to zero [21]. For example, if the colour value is 119 (01110111**1**), which is an odd number, the last bit is set to zero (01110111**0** =118). Once the image has been encoded with the exfiltrated data, a similar process must be run in reverse to get the information out of the image and write it back onto the disk as a file. In this method, the sender first selects the file to be exfiltrated and an image file from the disk. The image file could be in any image file format. However, the output file with the hidden data will be in the PNG format. Then, the sender compares the amount of the data that will be encoded into the image with the image dimensions because this method can only encode the amount of information that the image can hold. After that, the presented method is used to encode the exfiltrated file into the image. The selected image is opened and manipulated using the C2V library. As shown in Fig. 2, the input and output images are indistinguishable from each other by the human eye. The output image has a CSV file with 50,000 rows of data and 5.44MB in size, encoded in it. The input image is a PNG image of 26.1MB and the output was only 25.6 MB. Using a zip file will help to reduce the size of the file, and in turn, reduces the time taken by the method to transmit or encoded the data. For reassembling the encoded data, the receiver reads the last bit of every colour in every pixel into a file.

Fig. 2. Input (left) and output (right) images

2) Header-based methods: Sometimes sensitive information is concealed inside a packet header or email header and exfiltrated from the network of an organisation without being noticed [28]. This could be done by changing the expected values to encoded bytes of data. Several methods have already exploited parts of the packet header to hide exfiltrated data such as TTL, IP address and MAC address [21].

Time to live (TTL): The IP TTL field can be used to hide exfiltrated information by adding the value of a byte to the preset TTL, which will always be the same [29]. This will allow the receiving component to subtract the fixed TTL value from the received value and then convert the result into a binary number for the byte. By reassembling the binary numbers gathered from the TTL fields into an array, a copy of the file will be produced and written to the disk. This method is limited to one byte of information per packet, which increases the amount of traffic being sent across the network. In this work, we implemented this method by utilising a base value of 64 and then, adding the decimal value of a byte from the exfiltrated file to the base value. With this, the TTL can vary from 64 up to 319. First, the file to be exfiltrated is converted to a binary array, then, each byte (8 bits) in the string is converted to a decimal number. After that, the sender crafts packets with spoofed source IP addresses and added a decimal number to each packet's TTL field in the same order. The file name and extension are then encoded into UTF-8 and converted to a decimal array, which is added to the default TTL of another set of crafted packets. At this step, the source address is spoofed as the second address given by the sender. Once all the packets have been created, they are sent through to the destination address the sender has provided. A slowing function is used to add random deviation to the times that the packets are sent, ensuring so that there is no noticeable jump in the network traffic of the sending machine, which may otherwise draw unwanted attention. The receiving machine should know the two spoofed IP source addresses in order to listen for traffic coming from these addresses. Once all the packets successfully received, it collects all the TTL values in two arrays by source

IP address. The content array is converted to a binary file, while the name array is decoded into plain text.

IP options field : Exfiltrated data can also be hidden within the options field of the TCP/IP protocols as some of them do not have restrictions on what can be included in this field [21]. This method has faster speed than other header-based methods and not limited to any size restriction of the amount of data being sent in the packet, which reduces the amount of traffic being sent across the network, which, in turn, reduces the chances of this method being detected. In this work, we selected a pre-existing, unrestricted options field because, in the testing, we found that the custom-length option (maximum 20 bytes) could become malformed rather frequently. Precisely, we used the "RFC3692-style Experiment" [30], which has a maximum length of 20 bytes. First, the sender selects the file to be exfiltrated and convert it to a hexadecimal array. While the name and extension of the file are encoded into ASCII and then hexadecimal. After that, crafted packets are used to encode the hexadecimal data in the specified 20 bytes. Once all necessary packets are generated, they all sent to the receiving machine using the predefined delay by the sender. Like the previous method, two spoofed IP addresses are used to separate the content from the file name and extension.

Email Header: Stolen information can also be stored in the email header without being noticed [31]. In this case, data are usually encoded using encryption algorithms to avoid detection by discerning users. Other tactics convert the exfiltrated data to other formats, such as MAC addresses, which would reveal incorrect information to security investigators. In this work, we have set up a google account, where the sending machine can use it through Google account security. Initially, the sending machine selects the location file to be exfiltrated, the receiver's email address and a subject line. The subject line must be unique as it will be used by the receiver to retrieve the emails and reassemble the exfiltrated data. Then, the sender, converts the binary array of the file to be exfiltrated into an array of two hexadecimal values, which in turn, is converted into number chunks. This makes them look like MAC for concealment and avoiding detection. The MAC addresses are added into the headers of emails, which are then sent to the receiver with the subject line given by the sender. The receiving machine should know the email login of the receiving email account and the subject line of the emails sent with the hidden data. It contacts the email server and retrieves all the emails that match the given information. Then the data is extracted from the emails and reassembled by using the timestamps on the emails to determine the correct order. The receiving and decoding parts only works with Gmail accounts. However, in the future, this could be enhanced by adding other account providers, or a self-hosted email server, simply by changing the specified SMTP and IMAP setting to match the provider.

3) Metadata-based methods: In this category, the data is not hidden inside the packet sections. One common example of such methods is the usage of timing control, where the time between packets being received indicates what has been encoded using a predefined schema [21]. The fact that this time is not constant, as packets could be routed through different paths, could be exploited to set the time delays when packets have been sent. In this work, two different groups of time delays have been used. The shortest time between sending indicates that the data should be read as a zero, whereas the longest gaps should be read as a one. So, the binary array of the exfiltrated file is first converted to a string, then, for each bit in the string, if it is one, a predefined delay is artificially added to sending the packet. The file name and extension are sent to the receiver in the first packet as a message. When the last bit is read from the string, a halt packet is sent as the last communication. This signals to the receiving machine that all parts of the data have been sent and analysis of the packets can begin.

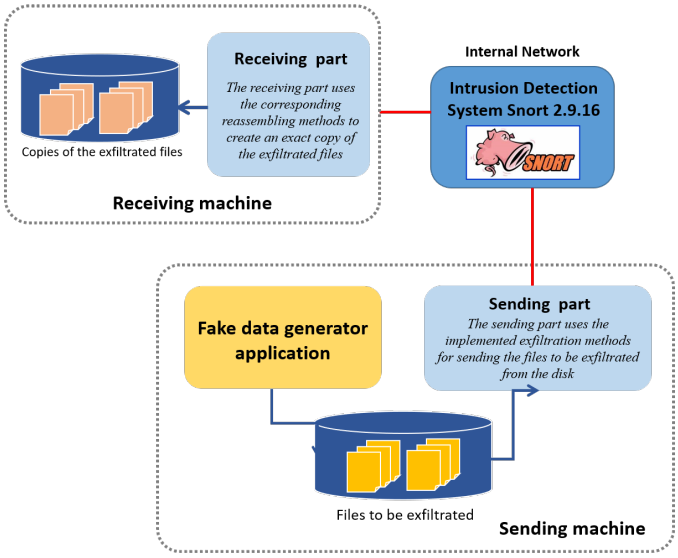

Fig. 3. Overview of the testbed

IV. Testing and Countermeasures

A. Test setup

In order to test the implemented data exfiltration methods, we have created a controlled experimental setup with two VMs on the same network; one for deploying the sending part and one for the receiving part. Another VM is used to implement the Network Intrusion Detection System (NIDS) Snort 2.9.16 [32], where the studied methods were tested against this NIDS. Snort is the most suitable signature-based NIDS for small networks [33]. It was chosen because it is open source, and most developers have experience in using and creating Snort rules. The snort's rule set configuration can be done with the default rules provided with Snort or downloaded from the Sourcefire Vulnerability Database (VDB) [34]. Users can also write their own rules that meet the specific needs of their networks [34]. After conducting intensive research, we did not find a good corpus with more realistic sensitive data that could be used for testing the implemented methods. Therefore, we have developed a python function that could generate files of any length and contains the following items: names, addresses;

date of birth; phone number; national insurance number; and credit card information including card number, security number and expiry date. Fig. 3 illustrates the implemented architecture and its components. The initial debugging, in all the implemented data exfiltration methods, was carried out with a small CSV file (1KB) generated by the fake data generator function. This will reduce the time required for the data transfer between the two devices and increases the speed at which the debugging could take place. In addition, any logic errors could be spotted on smaller files in a quicker time. The input and output files were hashed in order to test if they are identical. A larger CSV file was also used to test whether increasing the size of the input file would break the code in unexpected ways. In addition, a range of file formats was used to prove that all possible file formats will be accepted by the proposed methodology.

B. Testing results and countermeasures

The testing of the implemented methods for data exfiltration is done against the most stable release of Snort, which is Snort – 2.9.16. The tests were performed against the two freely available Snort community rule sets, which are available on the Snort.org web site. In Snort, a suspicious packet can be in one of two states: either "blocked" or "alerted". The blocked state indicates that the rules detected that Snort should stop all traffic. The alerted status is used to indicate that the packet may be suspicious and should be investigated further. Thus, in these tests, a method will be considered to have "Failed" the test, if it was blocked by the community rules. It will be considered "a partial success" if the method was only alerted to, as well as varying in success depending on how many of the packets are alerted. If the method passes the test without being blocked or alerted, it will be considered "a Full success".

Several tests were performed on the implemented methods for data exfiltration against the snort community rule sets. None of the exfiltration methods was blocked or alerted by the IDS and successfully evaded the current rule set of Snort. The images steganography method has been successfully implemented; however, it was excluded from these tests because it does not use network traffic.

In this paper, we have developed a new set of rules in order to enhance security against these methods. Table I presents the newly developed rules that were added to the Snort rule set and tested against the corresponding data exfiltration methods. The tests results showed that all the implemented methods, except timing control, were blocked and alerted by the newly created Snort rules. For the timing control method, the Snort IDS was not able to detect it as Snort only looks at packets in isolation and not as part of whole network traffic. So, this detection would have to be boosted up to the SIEM tools and would probably have to involve looking for something similar to beaconing. The basic prevention for this is to look for single device on the network that is attempting to call out the same destination over a long period of time, i.e., one day. Then if there are enough callouts within the given time frame the SIEM tool can flag them as suspicious.

Table I includes notes on relevant information about additional countermeasures that may need to be implemented in conjunction with Snort new rules. These would increase the accuracy of alerts by reducing the false-positive rates that the new rules will likely cause when applied in an enterprise setting with large and varied network traffic. Applying a SIEM tool, as described, would collect the snort logs and perform automated analyses on the logs.

TABLE I
PROPOSED DETECTION METHODS

Method	The new Snort rule	Additional countermeasures
HTTP 404 error	alert tcp any $->$ any any (content:" 404 Not Found"; width: 30; sid:3000001)	There is probably a need to add a SIEM tool to review the information generated from this, as this will flag on all 404 errors. The tool could look for multiple 404 errors from the same source address, to the same destination, over a period of time
Time to Live (TTL)	alert tcp any $->$ any any (ttl: ![0, 64, 128, 256]; sid:3000002)	By looking for non-standard TTL times, it may be possible to identify packets with information being sent through them. For better identification of these instances, a SIEM tool could be used to evaluate the number and consistency of the packets coming from each device on the network. That information can be used to highlight any abnormally large number of packets coming from a single device, as this would indicate this type of data exfiltration
IP options field	alert tcp any $->$ any any (ipopts: any; sid:3000003)	As this is an experimental options field, there should be a small number of false positives using this rule.
Email headers	alert tcp any $->$ any 25 (msg:"attempted use of port 25 for SMTP"; sid:3000004)	alert tcp any -¿ any 456 (msg:"attempted use of port 456 for SMTP"; sid:3000005) alert tcp any -¿ any 587 (msg:"attempted use of port 587 for SMTP"; sid:3000006) A simple web gateway block on all SMTP and IMTP except thoughts used by the organisation would stop any unseen exfiltration happening as organisations can log emails sent and received by internal email addresses

V. CONCLUSION

In this paper, we have implemented new methods that could be exploited by attackers to exfiltrate sensitive data from the internal network. This includes image steganography, HTTP 404 errors on web logs, TTL manipulation, IP options field manipulation, hiding data in email headers and timing control. The implemented methods were initially tested in a controlled environment against the community rule sets of the open-source IDS Snort, where all of them have been successfully evaded detection. The created testbed was also used to demonstrate how these methods operate. The deep understanding of how the exfiltration methods work, gained from the development and initial testing stages, was instrumental in constructing a new set of Snort rules that better detect the exfiltration of the data using the specified methods. Also, we

have proposed additional countermeasure that could be used in conjunction with the new rules to enhance the detection accuracy and reduce the false-positive alarms that the new rules could produce, especially, when applied in real-world applications. The tests results showed that the newly created Snort rules are able to block and alert all the implemented methods, except the timing control method. However, some countermeasures have been proposed in order to detect and alert this method for data exfiltration.

Image steganography has been successfully implemented as described in their original designs; however, it has not been tested against the Snort rule sets because it does not use network traffic. In this paper, we have only demonstrated how this method operates. In future work, we plan to propose prevention and detection countermeasures for this method. It would also be interesting to study the meta-type methods further. Combining all of the header methods into one single run on packets could bring advantages, as doing this with the IP options and TTL methods was found to increase the speed by reducing the number of packets to be sent for the file size. Finally, we hope that the proposed methodology can aid IT professionals to bring the issue of data exfiltration to the forefront of cybersecurity planning and actions within businesses.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 833673. The work reflects only the authors' view and the Agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] F. Ullah, M. Edwards, R. Ramdhany, R. Chitchyan, M. A. Babar, and A. Rashid, "Data exfiltration: A review of external attack vectors and countermeasures," *Journal of Network and Computer Applications*, vol. 101, pp. 18–54, 2018.

[2] A. Shabtai, Y. Elovici, and L. Rokach, *A survey of data leakage detection and prevention solutions*. Springer Science & Business Media, 2012.

[3] S. Alneyadi, E. Sithirasenan, and V. Muthukkumarasamy, "A survey on data leakage prevention systems," *Journal of Network and Computer Applications*, vol. 62, pp. 137–152, 2016.

[4] A. Giani, V. H. Berk, and G. V. Cybenko, "Data exfiltration and covert channels," in *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V*, vol. 6201. International Society for Optics and Photonics, 2006, p. 620103.

[5] TESSIAN, "The state of dlp 2020," tESSIAN, url=shorturl.at/tIU27, (accessed: Nov. 26, 2020).

[6] Foley, "Research revealed that up to 98% of iot devices are vulnerable and can be exploited in data breaches," cYCLONIS, 2020. url=shorturl.at/novJM, (accessed: Dec. 19, 2020).

[7] C. J. D'Orazio, K.-K. R. Choo, and L. T. Yang, "Data exfiltration from internet of things devices: ios devices as case studies," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 524–535, 2016.

[8] P. Gordon, "Data leakage-threats and mitigation," *InfoSec Reading Room*, 2007.

[9] D. A. Haddon and H. Alkhateeb, "Investigating data exfiltration in dns over https queries," in *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*. IEEE, 2019, pp. 212–212.

[10] C. Alcaraz, G. Bernieri, F. Pascucci, J. Lopez, and R. Setola, "Covert channels-based stealth attacks in industry 4.0," *IEEE Systems Journal*, vol. 13, no. 4, pp. 3980–3988, 2019.

[11] X. Liao, Y. Yu, B. Li, Z. Li, and Z. Qin, "A new payload partition strategy in color image steganography," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 3, pp. 685–696, 2019.

[12] K. Sudeepa, K. Raju, K. Ranjan, and A. Ghanesh, "A new approach for video steganography based on randomization and parallelization," *Procedia Computer Science*, vol. 78, pp. 483–490, 2016.

[13] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[14] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander, "The new threats of information hiding: The road ahead," *IT professional*, vol. 20, no. 3, pp. 31–39, 2018.

[15] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–26, 2015.

[16] G. Acar, S. Englehardt, and A. Narayanan, "No boundaries: data exfiltration by third parties embedded on web pages," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, pp. 220–238, 2020.

[17] J. Steadman and S. Scott-Hayward, "Dnsxd: Detecting data exfiltration over dns," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–6.

[18] Y. Liu, C. Corbett, K. Chiang, R. Archibald, B. Mukherjee, and D. Ghosal, "Sidd: A framework for detecting sensitive data exfiltration by an insider attack," in *2009 42nd Hawaii international conference on system sciences*. IEEE, 2009, pp. 1–10.

[19] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," *Computers & Security*, vol. 80, pp. 36–53, 2019.

[20] M. Schmidt, S. Fahl, R. Schwarzkopf, and B. Freisleben, "Trustbox: A security architecture for preventing data breaches," in *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*. IEEE, 2011, pp. 635–639.

[21] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM systems journal*, vol. 35, no. 3.4, pp. 313–336, 1996.

[22] E. Zielińska, W. Mazurczyk, and K. Szczypiorski, "Trends in steganography," *Communications of the ACM*, vol. 57, no. 3, pp. 86–95, 2014.

[23] M. Mahdi Hashim and M. S. MOHD RAHIM, "Image steganography based on odd/even pixels distribution scheme and two parameters random function." *Journal of Theoretical & Applied Information Technology*, vol. 95, no. 22, 2017.

[24] C. Francis-Christie and D. Lo, "A combination of active and passive video steganalysis to fight sensitive data exfiltration through online video," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 371–376.

[25] S. J. Stolfo, M. B. Salem, and A. D. Keromytis, "Fog computing: Mitigating insider data theft attacks in the cloud," in *2012 IEEE symposium on security and privacy workshops*. IEEE, 2012, pp. 125–128.

[26] A. R. Vinod, B. S. Sunidatta, K. U. Rani, and P. P. Sasidharan, "Hindering data theft attacks through fog computing," *International Journal of Research in Engineering and Technology*, vol. 3, no. 09, pp. 427–429, 2014.

[27] AWS, "Working with amazon s3 buckets," aWS, 2020. url=shorturl.at/etCHP, (accessed: Dec. 15, 2020).

[28] K. Ahsan and D. Kundur, "Practical data hiding in tcp/ip," in *Proc. Workshop on Multimedia Security at ACM Multimedia*, vol. 2, no. 7, 2002, pp. 1–8.

[29] S. Zander, G. Armitage, and P. Branch, "An empirical evaluation of ip time to live covert channels," in *2007 15th IEEE International Conference on Networks*. IEEE, 2007, pp. 42–47.

[30] T. Narten *et al.*, "Assigning experimental and testing numbers considered useful," BCP 82, RFC 3692, January, Tech. Rep., 2004.

[31] V. Caesar and R. Munir, "Modified email header steganography using lzw compression algorithm," in *Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019)*. Atlantis Press, 2020, pp. 104–112.

[32] Snort, "Snort," snort,2020. url=https://www.snort.org/, (accessed: Dec. 21, 2020).

[33] F. Alsakran, G. Bendiab, S. Shiaeles, and N. Kolokotronis, "Intrusion detection systems for smart home iot devices: experimental comparison study," in *International Symposium on Security in Computing and Communication*. Springer, 2019, pp. 87–98.

[34] CISCO, "Cisco vulnerability database library for firepower system,," cISCO,2020. url=shorturl.at/ikrDN, (accessed: Dec. 13, 2020).