Anonymized Network Monitoring for Intrusion Detection Systems

D B Srinivas[1†] and Sagar Mohan[2††]

srinivas.db@nmit.ac.in, sagarmohan.dev@gmail.com

Department of Information Science and Engineering
Nitte Meenakshi Institute of Technology, Bangalore, India.

Summary
With the ever-increasing frequency of public sector and small-scale industries going live on the internet in developing countries, their security of which, while crucial, is often overlooked in most cases. This is especially true in Government services, whilst essential, are poorly monitored if at all. This is due to lack of funds and personnel. Most available software which can help these organizations monitor their services are either expensive or very outdated. Thus, there is a need for any developing country to develop a networking monitoring system. However, developing a network monitoring system is still a challenge and expensive and out sourcing network monitoring system to third party is a security threat. Therefore, in this article we propose a method to anonymize network logs and outsource networking monitoring system to third-party without breach in integrity of their network logs.

Keywords:
Django, TShark, Computer-networks, intrusion-detection, anonymization

1. Introduction

Modern networking has become rather convoluted, not from a technical viewpoint, but from a managerial perspective. There are so many bits and bobs that go into a modern computer network, most people who are not actively involved in the field or have prerequisite knowledge in it are generally lost. This is evident in the state of Public Sector (Government) technology infrastructure and that of small-scale industries in across the globe; poorly maintained websites running on ancient/ under-powered, many-a-times misconfigured or outdated servers. While the central government has taken the reins and is now permeating guidelines and standards [1] [2] [3] to be followed by all, this is still not adopted by many organizations. Meanwhile, when it comes to monitoring these networks, most of the time it is not even taken into consideration, leaving to large deficits in security, especially in India [4].

In recent past, there have been multiple lapses, though not completely unavoidable, in security. The chief of them being when, in 2021, foreign malicious actors comprised the two-factor authentication system used by the Indian government [5]. The Covid-19 Pandemic did no favors as it took all work and related things online, cyber attacks rose 300% in 2020 [5]. The government however has acted and trained quite a few officials in cyber security [6]. There are many areas of improvement, even within the security policies themselves [7], however, the key concern is with network security itself.

Internationally, there have been multiple network breach incidents as well, notable recent ones such as, in February 2022 [8], The Red Cross had their networks compromised by an alleged state-backed hacker group. The attack included specific use of self-made tools to breach their network, evading their in-place anti-malware tools. It was only after installing observer agents in their edge systems that they were able to determine that their networks had been compromised. In another case, which occurred in May 2022 [9], hackers breached Greenland's healthcare systems causing extended wait times leading to significant delays in the working of the country's healthcare system.

Speaking of network security, this is usually done by always observing the traffic through the use of network capturing devices and software. Every packet flow which goes through an observation point is captured and logged into a database of sorts, after which a software makes sense of the packets by aggregating the information to a more presentable form, such as an end-user dashboard. However, due to the nature of these software products, the data presented to the user is usually, though encrypted in transit, is in plaintext. Whilst this is completely normal if the user is part of that organization, what if the user was not. Such a software could completely abstract sensitive information from the end-user and still provide functionality to a considerable extent. Thus paving the way for having anyone monitors the network without having to worry about data theft or data integrity loss.

In this paper, we present such a solution. Utilizing a

Manuscript received July 5, 2022
Manuscript revised July 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.7.23

web dashboard linked to a backend through a web API. We chose against making a standalone software as that would imply sinking time and effort in hardening the software itself; on the other hand, web browser security has come a long way and thus give us a good foundation to create a web dashboard whilst maintaining a reasonably good amount of security.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 describes implementation of proposed framework. In Section 4, process of anonymization is presented. In Section 5, performance analysis of our proposed framework is discussed. Finally, in Section 6 we conclude the paper.

2. Related work

In this section, we study the background information related to the anonymization of network logs for Intrusion Detection System (IDS) system. We study the existing system under the following categories.

A. ***Intrusion Detection Systems***

Computer Networks, like all types of distribution networks, are susceptible to many kinds of attacks, one of which is prevalent is the intrusion attack. By definition, intrusion can be from of unauthorized access to a network or system that is connected to a network. This type of attack can lead to complete integrity loss of the information in transit through the network. An IDS can be broadly classified into 2 categories [10]

- ***Active IDS***
  This kind of IDS acts as the middleman between the networking hardware and the information network. All information packets in transit through the network has to go through these IDS before it can leave its host network. This is done through the use of a hardware device which captures the network flow.
- ***Passive IDS***
  This kind of IDS does not require it to be a middle man in the network, as it can passively monitor network flow using a mirror port in the networking hardware or network gateway. Hence, the packets can freely flow without having to go through an extra bit of hardware. Furthermore, the IDS can be classified into 4 types [10]:
- ***Network Intrusion Detection System (NIDS)***
  As the name suggests, this type of IDS works at the network level, monitoring flows at a router, gateway level
- ***Host-Based Intrusion Detection System (HIDS)***
  Here, this type of IDS is installed, usually, as a software component in each device that is meant to be monitored Here, software programs called "agents" are installed in workstations or other systems and can be made to store logs or trigger any alerts when required to do so.
- ***Signature-Based Intrusion Detection System (SIDS)***
  This is the most popular form of intrusion detection technique. It is achieved by utilizing a known threat vector, be it a malicious IP address, or a malicious payload and creating a signature of this known threat vector. Once a signature is created of this threat vector, this signature, which will in a way that uniquely identifies a particular threat, can be added to private or public databases of known signatures. This signature can then be used to identify any similar threat that might crop up.

- **Anomaly-Based Intrusion Detection System(AIDS)**

Here, the intrusion detection system performs it purpose, by analyzing the performance of the network over time and creating a baseline for it. Hence, if there is any suspicious activity, where there is more or less network flow than expected, the AIDS will alert the analyst to this anomalous behavior, hence the name

B. ***How effective are Host Based Intrusion Detection Systems***

Time and again, many researchers have said that signature-based IDS cannot effectively identity new attacks or attacks involving protocols and attack vectors not identified by the detection system [11]. However, what they fail to note is the use case and the environment the IDS will be run in. If an adversary is proficient and well-funded enough to target a small organization, then the organization in question should have a large budget to handle all their security detail. Therefore, the signature-based IDS still stand pretty well against most forms of network intrusions and therefore can be utilized relatively easily.

C. ***Packet Filtering***

In [12], the authors demonstrate the different kinds of identifiers that are important to correctly flagging an offending network packet. They compare whether, when filtering packets, does the fields being monitored or does the anonymization algorithm matter for correctly flagging the malicious packets. In their results, they were able to ascertain that it is indeed the fields that matter more than the algorithm. The authors used the Snort tool to compare results between the fields and the algorithm and found that a few fields were responsible for the most false positive results; they were:

- TCP Destination Port
- TCP Source Port
- IPv4 Source IP
- IPv4 Destination IP
- Timestamp of Packet
- Acknowledgement Number of TCP Packet

D. *Anonymization*

Anonymization of network logs have been highly researched field where newer ways of anonymizing logs have been brought forward regularly. The most well-known paper on this [13], firstly mentions how this kind of anonymization depends on the policies of the organization to which the network logs belong to. While the easiest effort would be, to completely remove the fields which have no relevance to the required use case, in the case of an intrusion detection system, dropping any fields would result in erroneous results which might impact the efficiency of the IDS significantly. Therefore, based on the policy, a uniform method of anonymizing network traffic needs to be utilized. This form of obfuscation needs to be cryptographically secure, and should not disrupt the efficiency of the IDS.

Furthermore, while anonymizing the data, one should take care that the anonymization techniques used should not be reproducible in any way whatsoever. This is a key factor, as if part of any identifying information is obfuscated, but the rest is left as is, it could give adversaries leverage to reconstruct the data from discernible data.

From Fig. 1, the understanding of the system working is ascertained.

1) The Watchdog/Observer performs the packet capture on the device it is installed on.
2) The captured packet information is sent to the filter to be analyzed whether it is malicious or not.
3) The relevant information is written to the CoreDB, which is the plain text database.
4) The anonymized data is written to the AnonDB.
5) Any non-essential packet information is written to MongoDB, which benefits from not having any schema to conform to.
6) The Dashboard which is viewable to the analyst is connected to only the AnonDB and, through an API, the MongoDB database. It shows the IP logs and any flagged malicious IPs.
7) The analyst can also use the API to alert the admin without having actual access to the CoreDB.

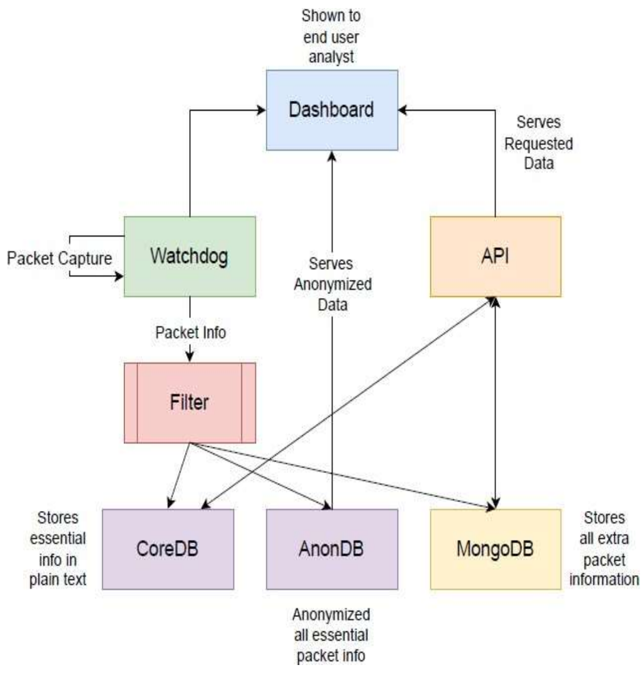
Shown to end user analyst
Dashboard
Serves Requested Data
Packet Capture
Watchdog
Serves Anonymized Data
API
Packet Info
Filter
Stores essential info in plain text
CoreDB
AnonDB
MongoDB
Stores all extra packet information
Anonymized all essential packet info

Fig. 1: Intrusion Detection System

E. *The Dashboard*

The Dashboard is built using the Django Web Framework. This is well defined in internet literature. Fig. 2 shows Django MVC Model architecture.

1) The Model: this uses the Object Relational Mapper to connect native code in Python to SQL statements. Therefore all database logic is written in native python code which allows for easier code reading and code modification.
2) The Template: Django provides the use of static templates along with template inheritance to allow for programmatic sequencing of existing templates.
3) The View: the view is the broker that ties together the model (the database) and the template and serves it to the requester

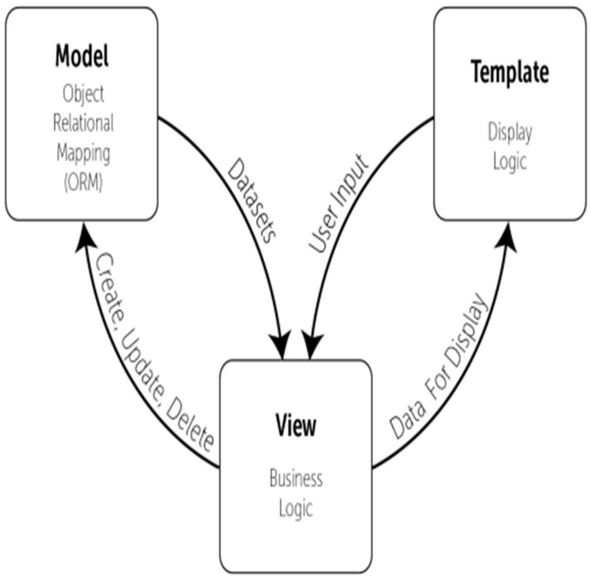

Fig. 2: Django MVC Model

*F. **The Observer***

The observer is an agent-based software that is supposed to be installed on the system or device meant to be monitored. This software is also written in python utilizing libraries such as Scapy, Database client libraries and the T-Shark Terminal Utility Program.

1) Tshark: Wireshark is a free and open-source program used for network packet analysis. It is used widely and thoroughly in the industry while being free software. Tshark is the terminal equivalent of the graphical user interface of the normal Wireshark software. Over all operation of Tshark is as shown in Fig. 3

2) How Tshark Works: Tshark, like most other packet monitoring software, works as follows:

- You bind tshark to listen to a particular network interface (this could be the wireless or Ethernet interface; it could also be a network device like a router as well).
- Now that tshark has been bound to an interface, when any application transmits its network data through this network, depending on what mode tshark is running in, it'll will capture the network data directly or indirectly.
- It performs the packet capture by either having the proper permissions to operate at a high privilege level in the operating system, or performing the packet capture with as little permissions as possible in user mode.
- When capturing the data, it captures all the network frames and contains it in a specific file format which is called the packet capture file format, aka pcap files.
- Any program utilizing this packet capture can either retroactively use the pcap files as they are being updated, or wait for the packet capture to complete, and the use the data contained within them.

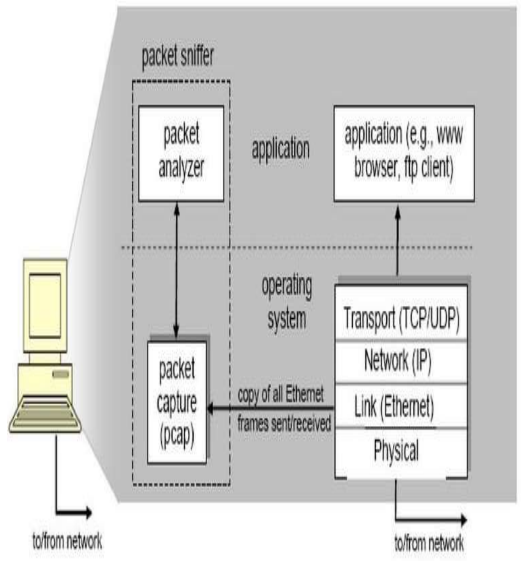

Fig. 3: TShark Operation

3. IMPLEMENTATION

The implementation of the observer is broken into four main parts. The packet capture; the IP filtering; anonymization; writing to the database.

*A. **Packet Capture***

Here, the interface which is supposed to be monitored is input into the config of the python script. After which the packet capture is invoked based on the time periods specified by the network administrator. Furthermore, the IP address of the current system is found to write the relevant data, whether a packet is incoming or outgoing. While the program is running, all the packet capture data is temporarily stored in a local directory to allow for log keeping. Since packet data is voluminous, it does not make sense to keep the entire raw packet data for long, as it gradually builds up.

B. *IP Filtering*

Once the data packet is captured, each relevant packet goes through filtering to check whether it is malicious or not. How this is done is by using the bloom filter [14]. The working of the bloom filter can be explained using the Fig. 4.

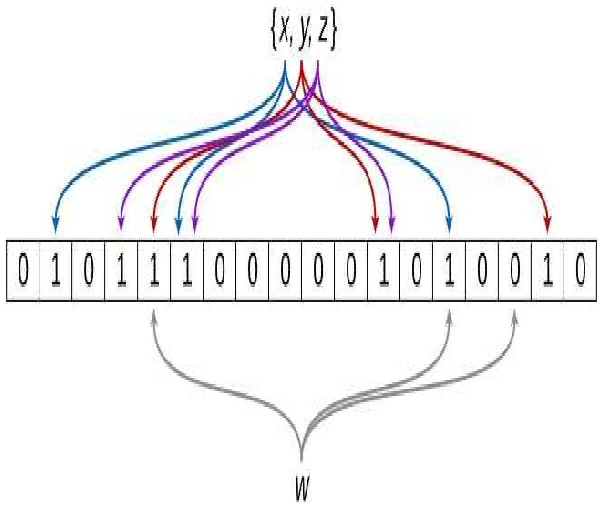

Fig. 4: Bloom Filter

.The Bloom filter data structure tells whether an element may be in a set, or definitely isn't. The only possible errors are false positives: a search for a nonexistent element can give an incorrect answer. With more elements in the filter, the error rate increases. Bloom filters are both fast and space-efficient. However, elements can only be added, not removed.

An empty Bloom filter is a bit array of m bits, all set to 0. There are also k different hash functions, each of which maps a set element to one of the m bit positions. Bloom filters are used in a many domains which include packet filtering, P2P communications, Big Data Security, to name a few [15]

From Fig. 4, the values *x, y, z* have been written to the bit array using some k hash functions. The values are converted into a set of bits that are written at certain indices in the bit array, in other words, the bits at those indices are *set*. When we try to check if the value *w* exists, we hash it and check if the bits at its indices are set, if they are, then *w* may be part of the set, if not set, it is not part of the set. Another example, let's assume that we need to add the words or names Jason and Davies to the filter. We can see that the two words are then mapped to particular indices in the bit array. Now a new word is to be searched in the bit array. The word "David" is hashed and check the bit array if the bits of the hashed word are set in the array. Since none of the indices of this new word are set, therefore the word "David" does not exist in the filter.

The IP filter works on the same principle. First, we obtain all major block-listed IPs from existing open-source information databases. Once we do that, we can add all those IP addresses to the bloom filter. We have to add each IP sequentially, as the hashing process hashes each IP and then sets the relevant bits in the bit array. When the observer wants to filter the packet logs, it pushes the information to the filter, which hashes the to-be-checked IP and sees whether the those bits are set or not. If all are set, it's definitely there, if some are set, it might be present, if none are set, it is not present.

When working with Bloom Filters the k number of hashes as well as the size of the set need to be predetermined. A smaller bloom filter will set all bits in the array to 1 if it fills up, so choosing an economical size is important. With hash functions, the more hash functions we use, the slower the bloom filter becomes, so a balance needs to be calculated between the filter size and the number of hashes.

From Fig.5, we can see the False Positivity Rate of the Bloom filter. Using one hash and writing 10 Million records into 100k size Bloom filter will make the FPR exactly 1, meaning all the bits are set to 1 so any record being searched will, by definition, be present in the array. With 8 hash functions and writing 10 Million records to a filter with a size of 1 Billion bits, the FPR comes down to $e - 10$.

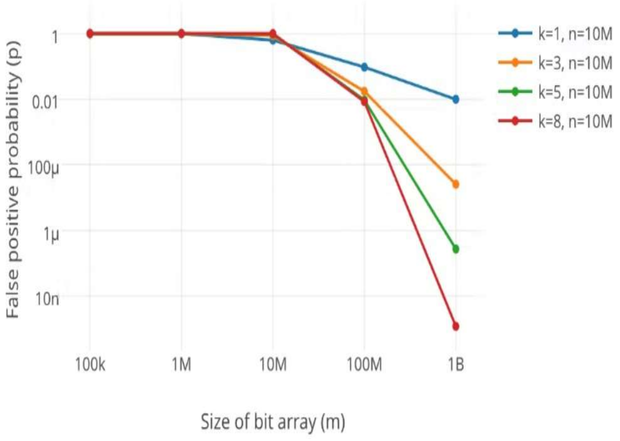

Fig. 5: Bloom Filter False Positivity Rate

C. *Anonymization*

Now, since the Observer has to do the anonymization locally, we have to implement the anonymization here. There are many ways one can go about implementing this, however let us first see the results of different ways of anonymizing the network traffic.

From Table I, we can see that randomization is the slowest, as it has to use the system entropy to generate each randomized entry. However, we can be rest assured that these slow, but randomized values cannot be brute forced as we see with the hashing algorithms.

D. ***Data Collected***

As per the framework design, data is predominantly stored in three databases, two of which are SQL and mirror the type of content stored but in two different ways. The *CoreDB* stores data in plaintext and is only viewable by an admin. The *AnonDB* is the database that mirrors most of which is stored in the CoreDB but all predetermined information is correctly anonymized. The data being stored is as follows

1) ***CoreDB and AnonDB:***

- Source IP
- Destination IP
- Source Port
- Destination Port
- Type of Service
- Universal Unique Identifier (UUID)
- Salt (if policy requires it, only stored in CoreDB)

2) ***Extraneous Packet Information:***

Packets contain many other information and depending on the type of service, different structures of the packet. All such information is directly fed into a NoSQL database, in our testing case, MongoDB. The reason for which being it is easier to write any other packet information into a NoSQL record rather than design a schema for every type of service's packet structure. For example, in the case of a normal TCP packet, the following information (out of many) is stored:

- Sequence ID
- Acknowledgement Number
- Window
- Checksum

4. *Anonymization*

Much research has gone into anonymization of network logs or PII (Personally Identifying Information), where the authors have created novel ways at effectively anonymizing network logs. When working with IPv4 Addresses, there are only 4 Million IP addresses available by design. Hence, considering this, if we look at brute-forcing the entire address set. It would take seconds to find a similar IP. We tested the time it would take to brute force the machine on two different machines resulting in similar times.

TABLE I: Hashing IP Addresses

Algorithm	1 Million IPs (seconds)	4 Million IPs (seconds)
Random	**115.53**	**493.77**
MD5	**2.35**	**8.33**
SHA1	**2.13**	**9.13**
SHA224	**2.35**	**9.60**
SHA256	**2.38**	**10.38**
SHA384	**2.72**	**11.05**
SHA512	**3.04**	**8.06**

From Table II, we can see that the time taken to brute force any of the Hashes on two machines, Machine 1 having an Intel Core i7-8750H @ 2.20GHz (this is a CPU variant for laptops) and Machine 2, having a desktop grade AMD Ryzen 7 3700X (16) @ 3.60GHz. This experiment was run on both machines using a single thread. A faster brute force time can be demonstrated when brute forcing voluminous IP addresses at scale, a great example is GPUHash which is online Cracking as a Service, although it is many a times used for nefarious purposes.

TABLE II: Brute Force IPv4 Addresses

Algorithm	Machine 1 (seconds)	Machine 2 (second)
MD5	**0.1847**	**0.0404**
SHA1	**0.2139**	**0.0436**
SHA224	**0.2595**	**0.5254**
SHA256	**0.2860**	**0.5459**
SHA384	**0.3967**	**0.0728**
SHA512	**0.1245**	**0.0219**

An alternative to this approach is to use a salt with an IP address to uniquely hash the IP address. However, when doing this, the storing of the additional salt field for each IP address as well as the final hash itself needs to be considered. As at scale, these two fields will accumulate much storage (in the orders for hundreds of gigabytes) and as such is not feasible.

Using completely randomized strings of characters representing anonymized IP logs are an easy and effective replacement to hashes in the system. Firstly, since the entire framework uses the concept of the UUID, the UUID can be utilized in searching for any unique record rather than relying on a unique hash generated using a salt. Secondly, since one can control the size of the randomly generated string, the storage space utilized will be much less than with Cryptographic hashes; for example, if using a SHA256 hash, it will generate a 256 bit or 32 byte unique signature for the given IP; the same if done using a string of 15 randomly generated characters takes less than half that space at 15 bytes.

Even with a completely randomized, utilizing system entropy, anonymization scheme, there is a minute chance of collision occurring, more specifically; taking a possible 50 character set and setting the generated string length at 15 character, there are 50^{15} number of different combinations of letters. So the possibility of a *birthday attack* is very unlikely. However, there is a more cryptographically secure alternative to this, the only downside being that the space needed to store records at scale increases. The solution would be to use *ShortUUID*s which are a more concise version of standard UUIDs. In the case of the implementation being used in this implementation, we used the Python library *shortuuid* which generates the UUIDs using Python's in-built *uuid* library and then converts into *base57* based on certain criteria. By design, they are 22 bytes long. Concatenating them further will cease to make them universally unique and no different from the randomized generation above.

5. Performance

Since the observer is running constantly to monitor network traffic. Some evident performance metrics play a major role.

A. *Large Capture Files*

The observer functions by using capture files taken by the capturing program, in this case, Tshark. So when there are large capture files with hundreds of thousands of packet capture information exceeding 200MB, then loading those files into memory will slow down the system.

B. *Packets Dropped*

When logging network packets in real time, more often than not, some packets get dropped. This is usually because the system is not designed to handle load. A few ways to prevent this or reduce this would be

- Stop or kill other programs which are using considerable amounts of memory
- Use a packet capturing program that is more designed for speed like TCPDump.

The framework is designed in such a way that any component or module can be swapped out without affecting other components.

C. *General Considerations*

In the case of the network observer software, there are two considerations that need to be addressed

1) Is the system dedicated?
2) If the system is dedicated for the observer, it should not pose any issues when capturing network traffic flowing through it as all computational power and memory is available to it
3) Is the System competent?
   When dealing with logging network traffic, the program should have good computing power, enough memory and not mention a good network interface card (NIC).

6. Conclusion and Future Work

Network security, especially that involving Intrusion Detection Systems, has a great scope for improvement. While there are much large-scale software to perform intrusion detection, what they lack is the correct set of anonymization potential to allow any 3rd party to monitor the network logs. Having completed this implementation, it has given us an in-depth understanding of the complexity of the networking flow that is always occurring unbeknownst to us, abstracted from us by decades of constantly progressing network engineering. The staggering number of packets that flows through a system is too fast for a human to comprehend now. As a result, the monitoring of such packets plays a paramount role in keeping computer networks safe. Additionally, the complexity behind well-known IDS software like IBM's QRadar, Splunk and us having done this project have made us realize the depth of programming that is required to monitor networks at a significantly large scale. The Network security domain is very exciting and we are thrilled to work forward and see where else something can be improved or worked upon.

While this implementation succeeds at anonymizing network logs securely and efficiently as possible, what it fails to do is allow analysis of encrypted network traffic, specifically the packet payload of each network traffic. A common way of skirting around this issue what be for the network packet capture software to act as the middle man and intercept any packets while not messing up any crucial packet info like the destination address. The problem, however, lies in the fact that not all applications will accept the certificates provided by the packet capture software. So an efficient solution to monitoring networks with least privileged access needs to be created.

REFERENCESS

[1] Ministry of Communications and Information Technology, Government of India, "National cyber security policy." [Online]. Available: https://www.meity.gov.in/writereaddata/files/ncsp 060411.pdf

[2] Ministry of Communications and Information Technology, Government of India , "Intermediary guidelines and digital media ethics code rules 2021." [Online]. Available: https://bit.ly/3AEjRxJ

[3] Ministry of Power, Government of India, "Cybersecurity in power sector guidelines 2021." [Online]. Available: https://bit.ly/3yUjjT1.

[4] S. Sharma and M. Khadke, "Network Security: A Major Challenge in India," in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*. IEEE, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8777642/

[5] N. Chauhan, "Almost 300% rise in cyberattacks in india in 2020, govt tells parliament," *The Hindustan Times*. [Online]. Available: https://www.hindustantimes.com/india-news/almost-300-rise-in-cyber-attacks-in-india-in-2020-govt-tells-parliament-101616496416988.html

[6] D. Bhardwaj, "In wake of increased attacks, govt trains 4,000 officials in cybersecurity," *The Hindustan Times*. [Online]. Available: https://www.hindustantimes.com/india-news/in-wake-of-increased-attacks-govt-trains-4-000-officials-in-cybersecurity-101617683297609.html.

[7] Col. Sanjeev Relia, "India's tryst with a new national cyber security policy: Here's what we need," Financial Express. [Online].Available: https://www.financialexpress.com/defence/indias-tryst-with-a-new-national-cyber-security-policy-heres-what-we-need/2304053/.

[8] S. Gatlan, "Red cross: State hackers breached our network using zoho bug," Bleeping Computer. [Online]. Avail¬able: https://www.bleepingcomputer.com/news/security/red-cross-state-hackers-breached-our-network-using-zoho-bug

[9] G. Cluley, "Greenland hit by cyber attack, finds its health service crippled," Bitdefender. [Online]. Avail¬able: https://www.bitdefender.com/blog/hotforsecurity/greenland-hit-by-cyber-attack-finds-its-health-service-crippled.

[10] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," vol. 2, no. 1, p. 20. [Online]. Available: https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7/

[11] E. Anthi, L. Williams, P. Burnap, and K. Jones, "A three-tiered intrusion detection system for industrial control systems," vol. 7, no. 1, p. tyab006. [Online]. Available: https://doi.org/10.1093/cybsec/tyab006

[12] K. Lakkaraju and A. Slagell, "Evaluating the utility of anonymized network traces for intrusion detection," in Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks - SecureComm '08. ACM Press, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1460877.14608999.

[13] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," vol. 36, no. 1, pp. 29–38 [Online]. Available: https://dl.acm.org/doi/10.1145/1111322.1111330.

[14] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," pp. 4047–4064, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128613003083.

[15] R. Patgiri, S. Nayak, and S. K. Borgohain, "Hunting the pertinency of bloom filter in computer networking and beyond: A survey," Journal of Computer Networks and Communications, vol. 2019, p. 2712417, Feb 2019. [Online]. Available: https://doi.org/10.1155/2019/2712417.

D.B. Srinivas received his PhD degree in Computer Science and Engineering from Visvesvaraya Technological University, India, in 2019. He is currently working as a Professor in Department of Information Science and Engineering at Nitte Meenakshi Institute of Technology, Bangalore, India. His research interests are in the area of Distributed Computing and network security.

Sagar Mohan is currently a Bachelor's student in the department of Information Science and Engineering at Nitte Meenakshi Institute of Technology, India. His research interests include computer networks, cyber security, and distributed systems.