ISSN: 1983 7402 ITA, 29 a 30 SET 2020

Evaluation of netfilter and eBPF/XDP to filter TCP flag-based probing attacks

Gustavo de Carvalho Bertoli[1], Lourenço A Pereira[2], Cesar Marcondes[2], Osamu Saotome[1]

Electronics and Computer Engineering - Aeronautics Institute of Technology (ITA)
[1]Electronic Devices and Systems (EEC-D) | [2]Computing (EEC-I)

***Abstract*— This paper presents a signature-based approach to secure networks by blocking TCP flag-based (Null, FIN, XMAS) probing attacks performed with the well-known Nmap security tool. Through packet filtering, this approach considers the deployment on Linux operating systems by low-level filtering through Linux Kernel Module (LKM) and Netfilter to directly operate at network stack. It also presents an alternative approach for packet filtering using the extended-Berkeley Packet Filter (eBPF) / eXpress Data Path (XDP) solution, which allows performing filtering at a lower level (network device driver), improving network filtering performance by 5% in comparison with the LKM/Netfilter solution. It also makes available an open-source baseline for packet filtering using both LKM/Netfilter and eBPF/XDP approaches.**

I. Introduction

Cybersecurity is a concern on a more connected world being a research topic with open challenges to be addressed. These open questions must be answered for the envisioned future to support applications such as connected vehicles, the internet of things, and autonomous systems. During systems attacks, the first required step is information gathering. As proposed by the cyber-kill chain [1], it is also called reconnaissance, which allows attackers to learn as much about their target without directly performing a more invasive attack. The data obtained in this phase allows the attacker to plan its next steps [2]. So, detect and interrupt an attack early, which prevents the success of the attack and saves costs and systems resources to perform defense.

The reconnaissance of a target aims to obtain information like available services (e.g., HTTP, FTP, SSH, RDP, Telnet) and network configuration. It is a very prolific field with service providers (e.g., shodan.io) that have crawlers probing the internet and providing a real-time database (Figure 1) of these findings with capabilities also to perform vulnerability testing on these mapped targets [3]. The availability of tools, computational resources, and even services raises awareness about probing attacks, reinforcing the importance of preventing them.

A common approach to prevent probing attacks is through packet filtering as Intrusion Detection Systems (IDS) like Snort [4] does. To perform packet filtering to detect probing attacks, the first method that can be used is through Linux Kernel Module (LKM) in conjunction with Netfilter. `netfilter` is responsible for the inner works of the well-known and widely deployed UNIX firewall called iptables[1]. The `netfilter` works with hooks on the kernel that allows the processing of network packets from specific points during its traversal path on Linux kernel [5].

The second approach is based on the eBPF/XDP (extended Berkeley Packet Filter / eXpress Data Path). BPF uses a Virtual Machine available in Linux Kernel that allows the execution of byte-code safely after a verification and validation process to avoid fault or security flaws. The extended version (starting on Kernel 3.18) increases the numbers of registers and instruction sets in 64 bits and a 512 bytes stack. The maps also allow interaction and data exchange with the application layer and tail-calls that address the limitation of 4096 bytes for eBPF applications. BPF is the inner work of the famous tcpdump application through the libpcap [6] [7]. XDP (available since kernel 4.8) allows BPF programs to run directly in the network driver - the network packet path's earliest point.

The comparison between the two approaches for packet filtering (LKM/netfilter vs. eBPF/XDP) is illustrated by the Figure 2.

Section 2 presents a Literature Review both on the probing attack and packet filtering. Section 3 presents the methodology adopted for this paper and the premises considered. Section 4 presents the results and discussions about the results obtained. Ultimately, Section 5 presents the conclusion about this study.

II. Literature Review

Regarding the TCP probing attacks, [8] presents the use of it to gather all Robotic Operating System (ROS)[2] deployments accessible through the internet. It makes use a tool called `ZMap` [9] to check for TCP port 11311 using TCP SYN packets. It also highlights the challenges of performing an internet-wide probing attack. The results of this internet-wide probing scan are the starting point to perform a security assessment on ROS. The next steps were to evaluate available ROS' topics and services and performed unauthorized remote control in the latter cases. It also provides recommendations to improve security, at minimum, [8] recommends reducing internet exposure.

For the packet filtering domain, [10] proposes an eBPF/XDP based firewall, in contrast to Netfilter/iptables, guaranteeing the same iptables semantics, connection tracking (stateful) and using another search algorithm instead of the linear search used by iptables. This proposed implementation outperforms iptables, especially when a large number of rules are required to be processed.

Gustavo de Carvalho Bertoli, gustavo.bertoli@ga.ita.br; Lourenço A Pereira, ljr@ita.br; Cesar Marcondes, cmarcondes@ita.br; Osamu Saotome, osaotome@ita.br.

[1]netfilter/iptables project - www.netfilter.org

[2]Robot Operating System - www.ros.org

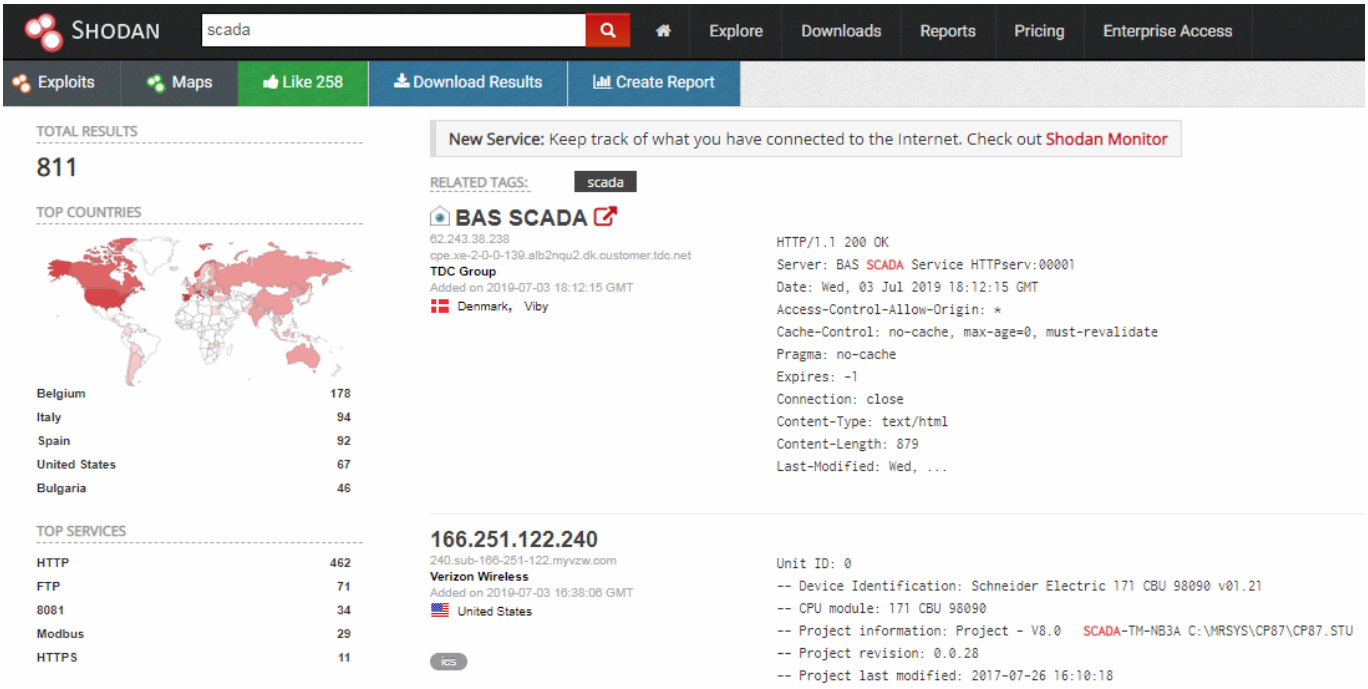

Fig. 1. Shodan search for ICS/SCADA

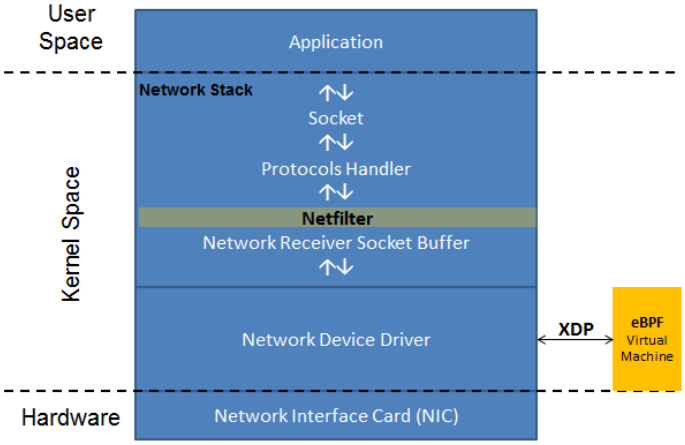

Fig. 2. XDP/eBPF and netfilter in the Network Packet Path

About the dataset for Intrusion Detection System (IDS) development, [11] presents a study, in their case, probing attacks and Denial of Service (DoS). First, it mentions the obsolescence of available and widely used datasets and presents an IDS dataset's properties:

- realism/representative
- validity (correct form packets)
- already labeled
- high variability
- correct implementation (attacks following standard)
- ease of updating
- reproducible
- no sensitive data

To compose the dataset with not malicious traffic, this normal traffic is generated considering random behavior and typical services being automatically labeled through IP addresses. Also, this automatic labeling is performed by considering malicious packets based on the attackers' source address.

It introduces three types of features for network IDS dataset, which are:

- header-based: relates to packets headers fields
- host-based: relates to communication between hosts (source and destination)
- service-based: relates to specific services communication between hosts (e.g. HTTP)

Then, [11] uses a machine learning approach for attack detection. First, it performs training in known attacks and evaluates algorithms' performance on scenarios of known, similar, and new attacks.

For the probing attacks, it is considered as known attacks the traditional TCP and UDP scans performed with `nmap` tool, as a similar attack, it uses the OS and service fingerprint also performed by `nmap`, and new attacks are those of vulnerability scan performed by Nessus[3].

It concludes that the machine learning approach has excellent performance for detecting known and similar attacks but does not have good performance for new attacks.

Finally, [11] dataset lack information about IP addresses for labeling purpose. Also, the lighter dataset (arff files) works with normalized values. Without more information about the mean and standard deviation of each attribute, it does not help to use the dataset to derive rules for embedded application, as this paper requires (lack of reproducibility).

[12] presents a detecting approach for probing attacks based on TCP protocol; it uses a stateless approach and machine learning (decision tree, naive Bayes, and KNN). It does not evaluate similar probing events or new events, as suggested and performed by [11]. [12] also highlights the importance of efficiency for security tasks when considered constrained devices, i.e., the Internet of Things. It also points out the difficulty of obtaining datasets for security research and proposes a dataset creation process. This process lacks representativeness

[3]Nessus: tenable.com/products/nessus

mainly when taking into account properties presented by [11]. This proposed dataset is used to perform training and tests and considers just as future work the implementation of the proposed packet filters, specifically kernel modules.

III. Methodology

The methodology adopted considers a stateless approach using only header-based features to evaluate each packet received from the network. Header-based features are those introduced by the Protocol to encapsulate data to be transmitted, such as:

- Ethernet-header features: length
- IP-header features: total length, time to live, protocol, checksum
- ICMP-header features: type, code, identifier, sequence number
- TCP-header features: sequence number, flags, ack number, window size, source and destination ports
- UDP-header features: length, checksum, source and destination ports

From the available headers' features, deterministic rules were created and tested to match packets transmitted during probing scan attacks. The probing attacks in this study's scope are the flag-based TCP port scanning techniques available through the widely known tool `nmap`.

The tests to confirm rule effectiveness were performed on `Wireshark` using a dataset containing multiple scan attacks and provided by [11] - specifically the known probing attacks dataset.[4] - and also locally using `nmap` and `Wireshark` in promiscuous mode to verify that the rules still apply in a live environment with the latest version of `nmap`.

Furthermore, the implementation of the rules was made both through Linux Kernel Module using `netfilter` and eBPF code using XDP deployment. Both implementations are them compared to evaluate effectiveness to reject `nmap` scan attempts and to evaluate network performance.

A. Probing Attack

Probing Attacks aims to evaluate the characteristics of the target. It is a reconnaissance step for a black-box approach of attack, which means that no prior knowledge about the target is available to attackers. Hence, a probing attack is usually the first step in an attack process.

In this paper `nmap` tool is used to perform these evaluations restricted only to TCP flag-based scan, but `nmap` also provides scans for UDP, ICMP protocols, and even application-level scanning. The characteristics that can be evaluated from the target are available services through port scanning.

1) TCP SYN Scan: TCP SYN scan is the most common scan due to its speed to be performed; on the other hand, most firewalls detect and prevent it. When not blocked, this probing provides clear differentiation between open, closed, and filtered as status for ports.
`nmap` syntax to perform TCP SYN Scan:

```
$ nmap -sS -v -n target_ip
```

Instead of performing the traditional three-way handshake from TCP, it sends an SYN packet from the attacker machine on this attack. After an SYN/ACK response from the target, an RST is sent from an attacker, closing the connection but aware that it is open. If closed, an RST packet is received from the target, and for a filtered case, no response is provided from target to attacker.

2) TCP Flag-based Scan: Flag-based scans exploit a TCP definition to define if a port is open or closed (RFC 793[5]). This aspect is that if the packet is sent without flags set SYN, ACK or RST, then an RST packet is sent for closed ports, nothing for open or filtered ports, and an ICMP packet for filtered ports [13].

These TCP flag-based probing attacks can circumvent non-stateful firewalls. Following attacks based on different TCP flags configuration is detailed:

XMAS Scan
Christmas (XMAS) scan set only FIN, PSH, and URG TCP flags.
`nmap` syntax to perform TCP XMAS Scan:

```
# nmap -sX -v -n target_ip
```

NULL Scan
NULL Scan do not set any TCP flags, so flag field is 0x00.
`nmap` syntax to perform TCP NULL Scan:

```
# nmap -sN -v -n target_ip
```

FIN Scan
For FIN scan just the FIN flag is set on TCP Flags.
`nmap` syntax to perform TCP FIN Scan:

```
# nmap -sF -v -n target_ip
```

An example of `nmap` output:

```
# nmap -sS localhost

  Starting Nmap (https://nmap.org)
  Nmap scan report for localhost (127.0.0.1)
  Host is up (0.000035s latency).
  Not shown: 996 closed ports

  PORT     STATE  SERVICE
  21/tcp   open   ftp
  23/tcp   open   telnet
  80/tcp   open   http
  631/tcp  open   ipp
  5432/tcp open   postgresql
```

B. Rules

Based on the presented `nmap` TCP probing attacks, characteristics of the transmitted packets by attacker were analyzed to evaluate useful attributes that allow the creation of rules to block these specifics attempts.

A first attribute that shall be analyzed by the filter is the flags field from TCP, once most of the considered probing attacks are based on this. Also through `Wireshark` inspection (Figure 3) and reference dataset [11] it is confirmed that `nmap` probing packets have a fixed window size of 1024 bytes.

Considering these aspects the generic rule is possible:

[4]TRAbID dataset: secplab.ppgia.pucpr.br/?q=trabid

[5]RFC 793: www.rfc-editor.org/rfc/rfc793.txt

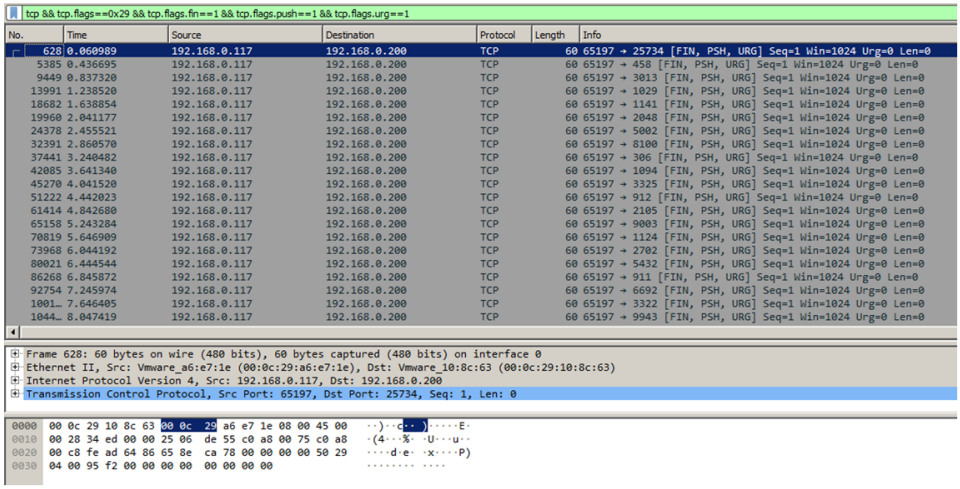

Fig. 3. Rule verification on [11] dataset using `Wireshark`

Listing 1
EXCERPT FROM LKM/NETFILTER IMPLEMENTATION TO BLOCK NMAP'S FLAG-BASED PROBING ATTACKS

```
if (tcph->window == htons(1024))
{
    if (tcph->fin == 1
        && tcph->psh == 1
        && tcph->urg == 1)
    {

        // XMAS SCAN
        return DROP;

    } else if (tcph->fin == 1
                && tcph->cwr == 0
                && tcph->ece == 0
                && tcph->urg == 0
                && tcph->ack == 0
                && tcph->psh == 0
                && tcph->rst==0
                && tcph->syn==0)
    {
        // FIN SCAN
        return DROP;

    } else if (tcph->fin == 0
                && tcph->cwr == 0
                && tcph->ece == 0
                && tcph->urg == 0
                && tcph->ack == 0
                && tcph->psh == 0
                && tcph->rst==0
                && tcph->syn==0)
    {

        // NULL SCAN
        return DROP;

    }
} else {
    return PASS;
}
```

C. Implementation

The rule presented in the previous section is easy to deploy on the baseline code of LKM/Netfilter and eBPF/XDP program, and both available at the GitHub repository [14].

For eBPF/XDP, it requires the LLVM/Clang compiler, and the load-unload of network interface shall be performed with `ip` Linux command. It is essential to highlight that eBPF/XDP requires a 64-bits Linux system (e.g., x86_64 or aarch64).

D. Performance Evaluation

The performance evaluation of both implementations are accomplished with `iperf3` [15] that is responsible to measure network throughput for each case, LKM/`netfilter` and eBPF/XDP approach. `iperf3` is used in the custom configuration that runs in a time-based evaluation of 10 seconds providing the average throughput. In conjunction with the average calculation performed by `iperf3`, the setup runs 20 times to avoid any discrepancies that can be generated by a single sampling.

`nmap` is used to confirm that the filtering rule is effective when both solutions are loaded, LKM/`netfilter`, or eBPF/XDP. The confirmation is when the filter is loaded, and no success is obtained with `nmap` attempts.

IV. RESULTS

Three configurations are considered for `iperf3` performance evaluation. The first configuration is the local computer without any packet filtering application labeled as none. The next configuration is the LKM/`netfilter` implementation, and finally is the network device driver with eBPF code loaded using the XDP.

Figure 4 presents a throughput increase of about 4% when comparing `netfilter` to eBPF/XDP approach. This analysis considers a simplified configuration of the local machine and Virtual Machine communication using virtual ethernet interfaces.

A second evaluation was performed between two personal computers, the host computer as `iperf3` server running on Linux Ubuntu 18 with Kernel 4.15 and the client computer running on MS Windows 7. The communication

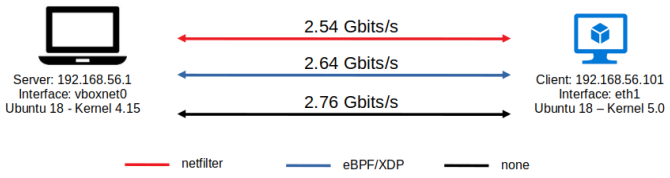

Fig. 4. Results from iperf3

was wired ethernet, and the performance evaluation confirmed the eBPF/XDP throughput increase of 5% compared to `netfilter` deployment.

Also, a local test was performed with `nmap` to guarantee that when the filter is loaded (`netfilter` or eBPF/XDP), the TCP probing attacks are ineffective.

```
drwxr@drwxr:~$ sudo nmap -sN localhost

Starting Nmap 7.60 ( https://nmap.org ) at 2019-07-04 21:46 -03
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:1723  ttl=37 id=52555 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:445  ttl=45 id=51774 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:113  ttl=50 id=29180 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:3306  ttl=52 id=39154 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:23  ttl=51 id=8113 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:995  ttl=52 id=36588 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:256  ttl=53 id=47954 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:80  ttl=49 id=64102 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:3389  ttl=47 id=42767 iplen=40  seq=3912854045 win=1024
sendto in send_ip_packet_sd: sendto(4, packet, 40, 0, 127.0.0.1, 16) => Operation not permitted
Offending packet: TCP 127.0.0.1:35840 > 127.0.0.1:139  ttl=56 id=31854 iplen=40  seq=3912854045 win=1024
Omitting future Sendto error messages now that 10 have been shown.  Use -d2 if you really want to see them.
Nmap scan report for localhost (127.0.0.1)
Host is up.
All 1000 scanned ports on localhost (127.0.0.1) are open|filtered

Nmap done: 1 IP address (1 host up) scanned in 201.42 seconds
drwxr@drwxr:~$
```

Fig. 5. Unsuccessful attempt to perform NULL Scan

As illustrated by Figure 5 - for TCP NULL Scan, these tests confirm the packet filter effectiveness to block the flag-probing attack. It is essential to highlight that the time required to perform the scan significantly increased when compared with `nmap` usage against a target that does not have a packet filter deployment.

V. Conclusion

This paper presented packet filters' effective use using `netfilter` through LKM and eBPF/XDP to prevent TCP flag-based probing attacks. It also compared the performance of both approaches with eBPF/XDP allowing higher throughput. Although it was not measured, it is crucial to highlight eBPF/XDP benefits of being faster and less computing consuming by not requiring the packet processing into the Linux kernel network stack compared to `netfilter`.

In this study, the rules implemented for filtering are deterministic and straightforward do not require complex implementation. It was not required for eBPF/XDP implementation to use Maps to interact with user application or tail-code for more extensive eBPF code deployment. On `netfilter` approach, no syscall was required to interface with the user application layer, and all data required for filtering was obtained through kernel network structures.

This paper provides an implementation baseline (LKM/-Netfilter and eBPF/XDP) for packet filtering purposes. Future works will perform a more rigorous performance evaluation, considering not just the iperf but also the kernel resources measurement (CPU and RAM usage). Also, to evaluate machine learning algorithms for packet filtering and its integration with eBPF/XDP implementation to take advantage of performance gains compared with traditional approaches. Moreover, for a more agnostic detection approach for probing attacks, the study has to consider other network protocols, attack tools, and techniques beyond the analyzed TCP flag-based probing attacks.

References

[1] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in *International Symposium on Security in Computing and Communication*. Springer, 2015, pp. 438–452.

[2] G. Weidman, *Penetration testing: a hands-on introduction to hacking*. No Starch Press, 2014.

[3] J. Matherly, "The complete guide to shodan: Collect. analyze. visualize," in *Make Internet Intelligence Work for You*. Leanpub, 2016.

[4] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[5] N. C. Team, "The netfilter.org project," 2001.

[6] M. A. M. Vieira, R. D. G. PacÃfico, M. S. Castanho, E. R. S. Santos, E. P. M. Camara Junior, and L. F. M. Vieira, "Processamento rápido de pacotes com ebpf e xdp," in *XXXVII Brazilian Symposium on Computer Networks and Distributed Systems, SBRC 2019, Gramado, Brazil, May 6-10, 2019*, 2019, pp. 1–50.

[7] Cilium, "Bpf and xdp reference guide," 2019. [Online]. Available: https://cilium.readthedocs.io/en/latest/bpf/

[8] N. DeMarinis, S. Tellex, V. Kemerlis, G. Konidaris, and R. Fonseca, "Scanning the internet for ros: A view of security in robotics research," *arXiv preprint arXiv:1808.03322*, 2018.

[9] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 605–620.

[10] S. Miano, M. Bertrone, F. Risso, M. V. Bernal, Y. Lu, and J. Pi, "Securing linux with a faster and scalable iptables," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 3, 2019.

[11] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, pp. 200–216, 2017.

[12] S. Barbieri, "Metodo para a deteccao de pacotes produzidos por scanning tcp," 2018. [Online]. Available: http://www.bdita.bibl.ita.br/tesesdigitais/lista_resumo.php?num_tese=75427

[13] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[14] G. d. C. Bertoli, "Netfilter and ebpf/xdp implementation of probing rule based detection," https://github.com/gubertoli/ids_ml/tree/master/src/probing_rule_based, 2019.

[15] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, "Iperf3: The tcp/udp bandwidth measurement tool," *URL https://iperf.fr*, 2005.