Signature-Based Detection of Botnet DDoS Attacks

Paweł Szynkiewicz(✉)

Research and Academic Computer Network (NASK),
ul. Kolska 12, 01-045 Warsaw, Poland
pawel.szynkiewicz@nask.pl
http://www.nask.pl

Abstract. The distributed denial of service (DDoS) attack is an attempt to disrupt the proper availability of a targeted server, service or network. The attack is achieved by corrupting or overwhelming the target's communications with a flood of malicious network traffic. In the current era of mass connectivity DDoS attacks emerge as one of the biggest threats, staidly causing greater collateral damage and heaving a negate impacting on the integral Internet Infrastructure. DDoS attacks come in a variety of types and schemes, they continue to evolve, steadily becoming more sophisticated and larger at scale. A close investigation of attack vectors and refining current security measures is required to efficiently mitigate new DDoS threats. The solution described in this article concerns a less explored variation of signature-based techniques for DDoS mitigation. The approach exploits one of the traits of modern DDoS attacks, the utilization of Packet generation algorithms (PGA) in the attack execution. Proposed method performs a fast, protocol-level detection of DDoS network packets and can easily be employed to provide an effective, supplementary protection against DDoS attacks.

Keywords: Network security · DDoS · Signatures · Detection · PGA · eBPF

1 Introduction

The detection of distributed denial-of-service (DDoS) attacks is a process that involves distinguishing malicious and normal network traffic in order to perform effective attack mitigation. The primary goal of a DDoS attack is to either limit or totally block the access to an application or network service, thereby denying legitimate users access to the service. Nowadays, an array of DDoS attack vectors has been recognized and studied [5,14,17]. Nonetheless, the principal of operation stays predominately the same i.e. to overwhelm targeted network resources with traffic coming from a mass of different sources. A seemingly simple problem of identifying and blocking specific malicious IP address proves to be a non-trivial task. Mainly due to the sheer distribution of attacking sources, which makes distinguishing the legitimate user traffic from attack traffic increasingly difficult, when spread across so many points of origin.

Consequently, original and more advanced DDoS mitigation techniques are being investigated. Solutions inspired by the control theory [7] or machine learning [3,4,12] that are able to recognize attack patterns in captured network traffic or aggregated traffic statistics are considered a next step in successful DDoS prevention. However, such methods have their limitations. Putting aside the costs and complexities of proper learning and tuning stages, for the pattern in network traffic to be recognized some amount of malicious traffic must pass through the protected network. In other words, DDoS can be detected only when the attack reaches proper scale inside the network. Solution described in this paper adopts the signature-based approach of DDoS detection. While this method lacks the self-learning aspect, signature-based DDoS detection methods can provide a effective, supplementary protection against DDoS attacks. Their main advantage being the ability to detect and block malicious network packets as soon as they arrive at the edge of protected network, practically mitigating the attack in its tracks.

In this paper we present our novel approach to signature-based detection of cyperthreats. PGA Filter is a prototype of a self-contained IDS system targeting botnet originating denial-of-service attacks by employing botnet fingerprinting techniques; packet generation algorithm (PGA) signatures. To our knowledge, this is the only complete ecosystem of this kind providing a full IDS experience. Our contribution includes definition of a new signature paradigm and description of a signature generation process. The core of the solution consists of the module responsible for the translation of signatures into packet filtering rules, and applying those rules to network traffic. The implantation leverages the enhanced Berkeley Packet Filter (eBPF) Linux kernel technology (since kernel ver. 4.9) that enables the augmentation of kernel logic with custom procedures. eBPF is currently considered a state-of-the-art for all packet processing requirements of networking solutions due to it's high efficiency and programmability. Our proof of concept study is finalized by the integration and tests of PGA Filter as a part of GUARD[1] cypersecurity framework.

The rest of the paper is organized as follows. In Sect. 2 the concept of PGA signatures is explained. We provide a brief understanding of the context required i.e. the principle of operation for network telescopes, the characteristics of botnets and botnet denial-of-service attacks and how we are able to identify them. Next, in Sect. 3 we describe the PGA Filter, its implementation and technologies employed. Finally, we give our conclusion in Sect. 4, which includes the overview of future plans.

2 PGA Signatures

Successful signature-based detection of DDoS attacks requires a source of specialized, high quality, up to date network traffic signatures. The proposed solution

[1] https://guard-project.eu/, Guarantee Reliability and trust for Digital service chains (GUARD).

integrates with the infrastructure built around the Network telescope (darknet) [2] developed under the SISSDEN project[2]. Network telescope provides the access to valuable and hard to come by data about ongoing mass-scale cyber-attack campaigns. By analyzing said data we are able to extract and generate specific, network packet level, DDoS attack signatures. The above-mentioned signatures, dubbed as PGA signatures advance a novel approach of depicting the patterns in malicious network packet headers. The idea of PGA signatures was previously investigated in SISSDEN project. The solution described in this paper builds on this knowledge to offer a prototype concept of PGA signatures defined by an explicit signature syntax.

2.1 Network Telescope

Network telescope (known also as a black hole, an Internet sink, darkspace, or a darknet) is an unused space of IP addresses that are used solely for the purpose of passive monitoring [2,16]. Unused IP addresses should receive no legitimate network traffic. In practice, however many packets are observed arriving at this IP space. This continuous view of anomalous unsolicited traffic is by definition classified as suspicious. This observed traffic is a result of a wide range of events, such as backscatter from randomly spoofed source denial-of-service attacks, scanning of address space by attackers or malware looking for vulnerable targets, the automated spread of Internet worms and viruses and various misconfigurations (e.g. mistyping an IP address or deprecated DNS records). In general, malicious activities observed in a form of victims' echo responses in darknet fall into following categories:

- backscatters from denial-of-service attacks,
- scanning activities,
- exploitation attempts.

Network telescopes are a valuable source of information about ongoing events in the whole Internet Infrastructure. However, access to this data is rather restricted. First of all, network telescopes are usually managed by the institution responsible for the administering of IP addresses for the given regions of the world. The list of unused IP addresses is a closely guarded secret, which should remain undisclosed to cyber-criminals. Also, there are serious privacy and security concerns associated with network telescope datasets. Viruses and worms may involve the installation of backdoors which provide unfettered access to infected computers. Therefore telescope data may inadvertently advertise these vulnerable machines. Additionally, while the source of some types of telescope traffic, including denial-of-service attacks and worms, is readily apparent, a significant volume of traffic is of unknown origin. Without identifying the causes of this traffic, the security and privacy impact of releasing these data cannot be categorically assessed.

[2] https://sissden.eu, Secure Information Sharing Sensor Delivery Event Network (SISSDEN).

For our purposes the access to network telescope data is granted thanks to the fact that NASK, as an institution governing the Polish IP address space, has established a darknet infrastructure. Network telescope at NASK is a time-tested, working system integrated with security incident exchange platforms (n6[3]) and actively used by Polish CERT. It passively observes and analyzes network traffic that reaches NASK sinkhole, which consists of over 250 thousands IPv4 addresses. Network telescopes at NASK is used internally by many cybersecurity teams and sends reports to external organizations, including Shadowserver[4]. All of the traffic reaching NASK darknet (around 10 000 network packets per second) is being captured and analyzed. On the basis of this data, it is possible to:

– collect intelligence regarding new threats and trends (also DDoS)
– detect both large-scale and targeted attacks,
– fingerprint actors responsible for those events.

2.2 DDoS Seen Through Network Telescope

Despite medium resolution of NASK's network telescope (resolution corresponds to the number of IP addresses the telescope monitors), it is "precise" enough to serve its purpose in our scenario, that is to monitor the spread of random-source distributed denial-of-service attacks.

To make it difficult for the attack target (and the target's ISPs) to block an incoming attack, the attacker may use a fake source IP address (similarly to a fake return address in traditional mail) in each network packet sent to the victim 1 as shown in Fig. 1.

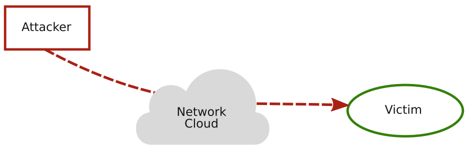

Fig. 1. The attacker sends packets with spoofed source addresses to the denial-of-service attack victim

This technique is known as IP spoofing in the trade. Because the victim of the denial-of-service attack can't distinguish between incoming requests from an attacker and legitimate inbound traffic, the victim tries to respond to every received request (see Fig. 2).

When the attacker spoofs a source address monitored by network telescope, we observe a response destined for a host that doesn't exist (and therefore could not sent the initial query), this effect is also know as a backscatter and is shown in Fig. 3.

[3] https://n6.cert.pl/en/.
[4] https://shadowserver.org.

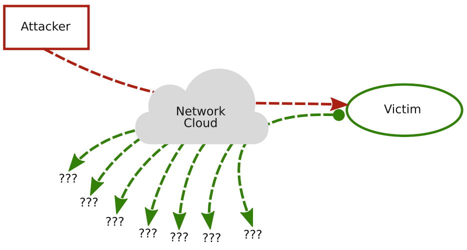

Fig. 2. The denial-of-service attack victim cannot differentiate between legitimate traffic and the attack packets, so the victim responds to as many of the attack packets as possible.

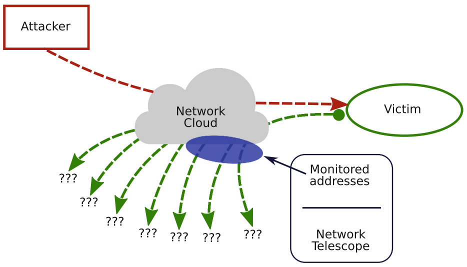

Fig. 3. Because the NASK's network telescope composes of 250 000 IPv4 addresses there is a chance that some of the responses to spoofed packets generated by the denial-of-service attack victim will be received.

By following these unsolicited responses, researchers can identify denial-of-service attack victims and infer information about the volume of the attack, the bandwidth of the victim, the location of the victim, and the types of services the attacker targets. Moreover, a deeper inspection of backscatter packets can lead to the discovery of certain characteristics and similarities between the values in the packet headers. This is the backbone of our approach.

2.3 Botnets

During recent years, traffic destined to darknet has been gradually evolving due to the growing numbers of rapidly expanding bot networks (botnets). The considerable impact of their operations is clearly mirrored by the network telescope observations; firstly through the presence of longer-duration, low-intensity events intended to establish and maintain botnets; secondly, by the dominating aftermath effects of well orchestrated, large-scale DDoS attacks.

Botnet [15,18,21] are collections of devices infected with a bot program which allows an attacker to control them. They range in size from only a few hundreds to millions of infected devices. Attackers typically use the collective resources of the botnet to perform various disruptive or criminal activities, such as sending vast amounts of spam emails, distributing malware and launching denial-of-Service attacks [6]. Botnets are considered one of the biggest threats to the Internet Infrastructure. The growing number of smart and connected but weakly secured and easy to compromise devices [9] (smartphones, IoT, etc.. . .) indicates that this threat will remain significant.

Network telescope proves a valuable asset in monitoring and disabling botnets [1, 13, 20]. Tracking of botnet activities is made possible through the process of fingerprinting. As described in the next section, the information required to identify (fingerprint) specific botnet packets can be extracted by analyzing network telescope data.

2.4 Packet Generation Algorithm

As mentioned in previous sections, botnets are mainly used by attackers to perform denial-of-service and scanning attacks. When the command is issued infected bots start to generate network traffic directed at the attack victims. The logic of the attack itself is usually an integral part of the bot malware. Malicious packets are constructed depending on the kind of attack performed. Often some additional logic is employed to speed up this process. The procedure behind packet generation is know as packet generation algorithm (PGA). Every PGA has some unique characteristics, packet generation rules are attack specific. Most relevant is also the fact that PGA are typically botnet specific, hence they can be used to identify the entity behind the attack (fingerprinting).

The malicious packets of botnet generated attacks usually include fixed patterns. Patterns occur when a single or multiple bytes of particular protocol specific parameters (e.g. TCP sequence number or IP destination address) are deterministically dependent on each other. A simple example of a fixed pattern is an equality of TCP sequence number and destination IP address, which is often observed during port scanning. In this case the PGA copies bytes from the address into the sequence number instead of generating a random one. The motivation behind this additional operations is to optimize computation complexity and memory usage of the algorithm. Botnet malware often targets devices with limited computing power (legacy PCs, IoT, etc.. . .), also botnet attack capabilities scale with the number of infected devices. Therefore every small improvement in packet generation speed greatly impacts the attack effectiveness.

Presented example of PGA for scanning purposes is of course a simple form of optimization. The authors of malicious software may include more sophisticated rules using logical shift, negations, incrementation, decrementation, bytes swapping etc. It all comes to a trade-off between computational simplicity and amount of effort required to detect a pattern.

2.5 Generating PGA Signatures

With the data provided by the darknet infrastructure, the main aim of the PGA analysis is the detection of packet generation rules in traffic generated by botnets and other malware. During online analysis of malicious network traffic the detection and documentation of packet generation rules used during particular network activity (e.g. TCP SYN scan or flood) is performed. The practical aspects of PGA signatures are described in [11,19]. Methods for extraction of PGA signatures candidates vary in nature. From the simplest, like deterministic conditions on header fields - PGA headers often defy RFCs guidelines, which makes resulting traffic stand out from legitimate one, to more sophisticated ones based on statistical, hierarchical analysis and machine learning. Constant monitoring of the incoming traffic by the network telescope helps maintain situational awareness with the proper collection of details about current threats and signatures for botnet fingerprinting.

The process of extracting PGA signatures from suspicious traffic involves reverse engineering of packet generation algorithm by analyzing the headers of packets arriving at darknet. First, packets sharing the same source and close arrival times are grouped together to establish a backscatter of possible DDoS attack (victims response to packets with spoofed source address). By inspecting such packets we are able to partially recreate the original DDoS packet that was sent to the victim (see Sect. 2.2). Finally, the most challenging part is determining whether recreated packets could be generated by the same packet generation algorithm, and if so, what are the steps and operations performed by said algorithm.

An example TCP/IP packet which could be a part of a DDoS backscatter observed by network telescope is shown in the Fig. 4. To recreate the original DDoS packet the following operations are performed:

1. Swap source and destination fields:
   IP: Source IP Address $\longleftrightarrow$ Destination IP Address
   TCP: Source TCP port number $\longleftrightarrow$ Destination TCP port number
2. Set the value of TCP Sequence Number to decremented TCP Acknowledgment Number

The result of those operations is presented in the Fig. 5, limited to only the key header fields. The A, B and C denote some specific byte values, in case of A and B a 4-byte value, and C a 1-byte value. It would be plausible that this example packet is a result of a packet generation algorithm. Such algorithm would reuse the first two bytes of source IP address, and copy it over to first two bytes of TCP sequence number. Next, the last two bytes of destination IP address would be copied over to the last two bytes of TCP sequence number. The signature of such PGA would state:

"The first two bytes (0, 1) of Source IP Address are equal to the first two bytes of TCP Sequence Number and the last two bytes (2, 3) of Destination IP Address are equal to the last two bytes of TCP Sequence Number"

or in our less verbose signature naming syntax:

```
ip-src:0:1_is_tcp-seq:0:1_and_ip-dst:2:3_is_tcp-seq:2:3
```

Presented principle of operation is part of semi-automated service running as a part of NASK's network telescope system. The algorithms are still being refined new reverse engineering methods are research. Currently supported internet protocols are: Ethernet, IPv4, IPv6, UDP, TCP, ICMP.

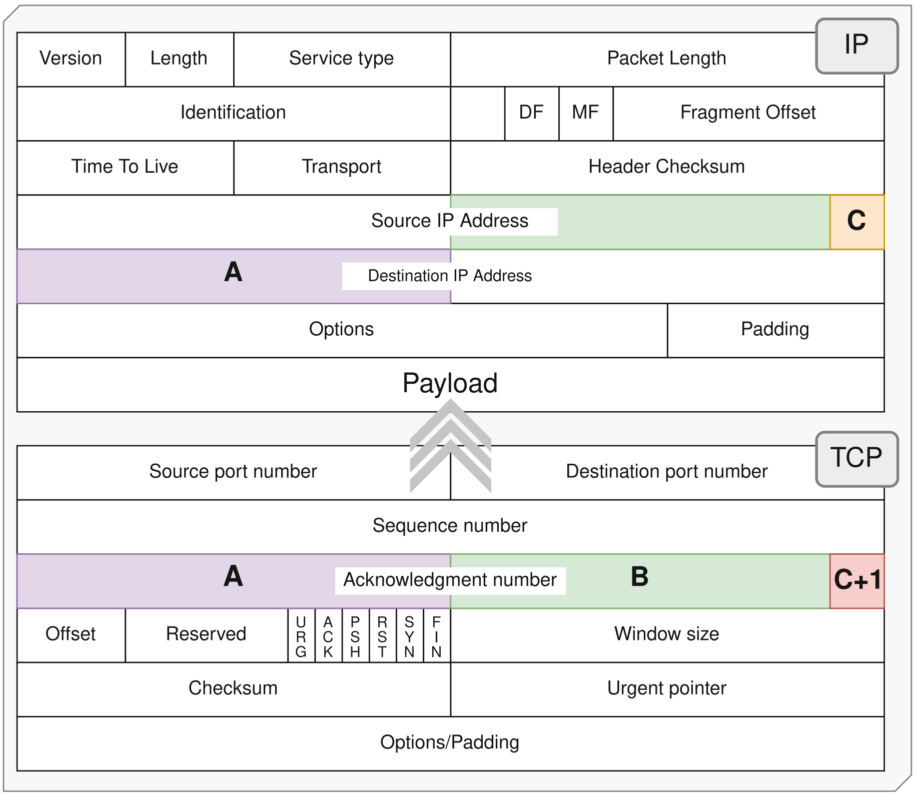

Fig. 4. *TCP/IP* packet observed by darknet.

3 PGA Filter

The main challenge is the implementation of mechanisms able to interpret said data to translate it into system-comprehendible set of rules, which in turn, can be deployed in client's network security infrastructure. Since the standard signature IDS/IPS solutions (Snort, Suricata, etc.) focus on analyzing the patterns in the payload data, rather than the dependencies between values in protocol headers, new solutions have to be developed.

3.1 Extended Barkeley Packet Filter

Proposed implementation makes use of eBPF (extended Berkeley Packet Filter), a robust, highly flexible and efficient virtual machine-like construct that allows for extending standard kernel in Unix-like systems with custom functionality by executing bytecode at various hook points in a safe manner. It is applicable in a number of Linux kernel subsystems, most prominently networking, tracing and security.

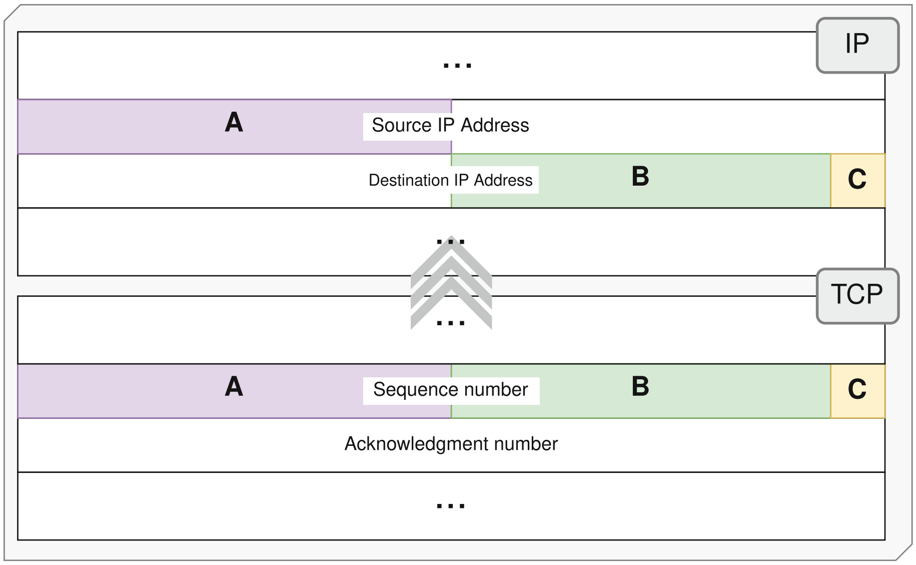

Fig. 5. Header fields of an original *TCP/IP* DDoS packet based on observed response packet.

BPF, as a concept, does not define itself by only providing its instruction set, but also by offering further infrastructure around it such as maps which act as efficient key/value stores, helper functions to interact with and leverage kernel functionality, tail calls for calling into other BPF programs, security hardening primitives, a pseudo file system for pinning objects (maps, programs), and infrastructure for allowing BPF to be offloaded, for example, to a network card.

The kernel subsystems making use of BPF are part of BPF's infrastructure. The subsystem utilized for the purposes of PGA signature rule-based detection is XDP (eXpress Data Path). XDP is a framework that makes it possible to perform high-speed packet processing within BPF applications. To enable faster response to network operations, XDP runs a BPF program as soon as possible, usually immediately as a packet is received by the network interface. However, since this processing occurs so early, full network stack is yet to be established, the packets' metadata extracted and parsed (Fig. 6).

BPF is a general purpose RISC instruction set and was originally designed for the purpose of writing programs in a subset of C which can be compiled into BPF instructions through a compiler back end (e.g. LLVM), so that the kernel can later on map them through an in-kernel JIT compiler into native opcodes for optimal execution performance inside the kernel. Among other use-cases, BPF as a technology, is a perfect candidate for the implementation of DDoS mitigation software. It provides the compromise between the efficiency of execution - inside the kernel, and the extensibility and programmability - subset of C programming language. Until recently similar results were possible only with custom compiled

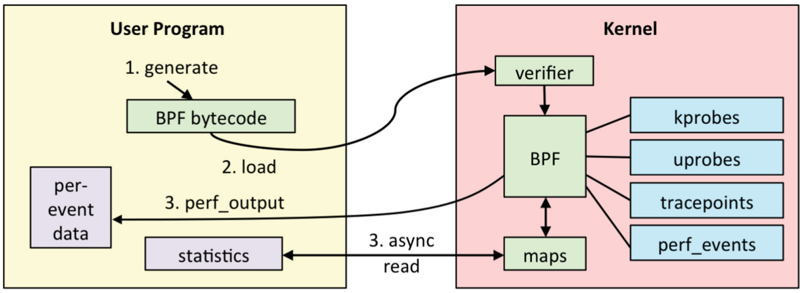

Fig. 6. *(e)BPF* lifecycle (https://www.brendangregg.com/).

kernels. Preparing a custom kernel is a cumbersome process, which requires a great deal of attention, as a simple mistakes can make the target system unstable. Understandably, such solutions are adopted with great deal of restraint and are rather unwelcome in third-party software. However, in contrast to custom kernel approach, BPF gives guarantees for safety and predictability of execution, also can easily be loaded and unloaded from the target system. By employing BPF as a main technology in our signature-based DDoS mitigation solution we are able to directly translate DDoS signatures in to sets of instructions executed inside system's network stack. The advantages for pushing these instructions directly into the kernel include[5]:

1. Making the kernel programmable without having to cross kernel/user space boundaries. For example, BPF programs related to networking, can implement flexible networking policies (firewall), load balancing and other means without having to move packets to user space and back into the kernel. State between BPF programs and kernel/user space can still be shared through maps whenever needed.
2. Given the flexibility of a programmable data path, programs can be heavily optimized for performance also by compiling out features that are not required for the use cases the program solves. For example program can support only required set of network protocols: TCP/IP, UDP, ICMP, etc.
3. In case of networking (e.g. XDP), BPF programs can be updated atomically without having to restart the kernel, system services or containers, and without traffic interruptions. Furthermore, any program state can also be maintained throughout updates via BPF maps. That means that programs logic (e.g. traffic filtering rule-set) can be modified on the go and hot-deployed without stopping the service.
4. BPF provides a stable ABI towards user space, and does not require any third party kernel modules. BPF is a core part of the Linux kernel that is shipped everywhere, and guarantees that existing BPF programs keep running with newer kernel versions. This guarantee is the same guarantee that the kernel provides for system calls with regard to user space applications. Moreover,

[5] Cilium project documentation (https://cilium.io/).

BPF programs are portable across different architectures, which makes our solution available for any up-to-date Linux systems.
5. BPF programs work in concert with the kernel, they make use of existing kernel infrastructure (e.g. drivers, netdevices, tunnels, protocol stack, sockets) and tooling (e.g. iproute2) as well as the safety guarantees which the kernel provides. Unlike kernel modules, BPF programs are verified through an in-kernel verifier in order to ensure that they cannot crash the kernel, always terminate, etc. XDP programs, for example, reuse the existing in-kernel drivers and operate on the provided DMA buffers containing the packet frames without exposing them or an entire driver to user space as in other models. Moreover, XDP programs reuse the existing stack instead of bypassing it. BPF can be considered a generic "glue code" to kernel facilities for crafting programs to solve specific use cases.

3.2 PGA Filter Architecture

PGA filter runs as a system daemon. The tasks performed by this daemon are:

- parsing configuration and rules,
- generating BPF bytecode based on configuration and rules,
- loading or unloading BPF bytecode to kernel,
- accumulating and sending packet statistics to Kafka.

The architecture of PGA Filter is presented in the Fig. 7. The whole implementation resides in *pgafitler.py* Python script responsible for parsing configuration, generating and loading of BPF code, gathering and sending results. The configuration is split between two files in YAML format: *rules.yml* and *config.yml* for convenience. The *rules.yml* file is dedicated for the declaration of available PGA signatures. It can be edited by hand by the administrator or downloaded from the signature sharing service. The general configuration resides in *config.yml* file. Besides the standard configuration parameters, i.e., the name of the inspected network interface or the desired logging level, this file contains the list of currently enabled PGA signatures from the *rules.yml* file.

The most crucial operation is translating chosen PGA signatures into valid BPF code. Generated BPF program performs packet inspection operations and gathers data regarding signature matches. The moment a new PGA filter configuration is applied, updated BPF program is generated and swapped with the previous one. During the update phrase, the BPF infrastructure ensures the atomic operation of swapping BPF code (unloading the previous program code and loading the new one). The inspection of network packets is performed continuously and without delay through the whole process.

Below, in Listings 1 and 2 two examples of configuration rules (in YAML format) corresponding to PGA signatures are presented. Each rule can be identified by a unique name (e.g. `''tcp-sport_is_tcp-seq:0:1''`) or numerical id (*sid* – signature id). Each rule also has a revision number (*rev*) which should be incremented every time said rule is modified. The *content* filed contains the

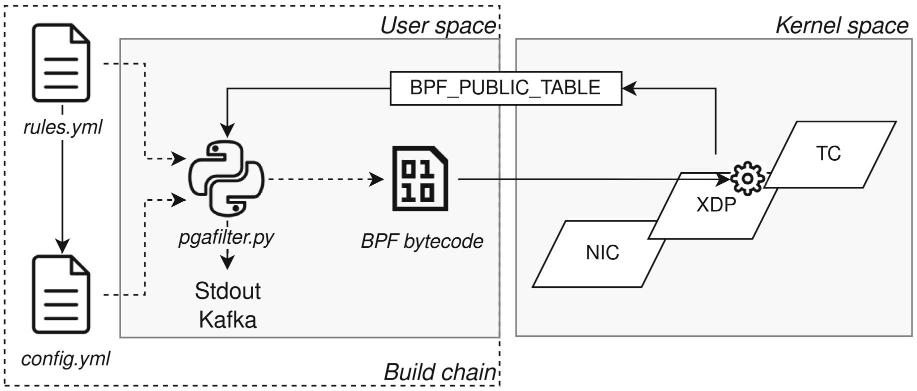

Fig. 7. Architecture of the PGA Filter framework.

representation of the PGA signature itself. In order to allow for high degree of flexibility, signatures are represented in valid C programming language syntax. Code can span across multiple lines of code separated by the semicolon (';'), but the last line must always evaluate to boolean (true or false) value. Standard syntax is extended with some helpful macros and function described bellow. The *args* filed declared which protocol headers should be passed to signature function. Multiple headers can be passed, each header corresponds to protocols supported by PGA filter, i.e.: ethhdr, iphdr, ip6hdr, udphdr, tcphdr, icmphdr. Headers are in fact structures defined in Linux Kernel User API source code [8,10].

```
"tcp-sport_is_tcp-seq:0:1":
    sid: 1003
    rev: 1
    args:
        - tcphdr
    content: "tcphdr->source == pga_2_byte32(tcphdr->
        seq, 0, 1)"
    comment: "TCP SOURCE PORT is equal to TCP SEQ
        bytes 0:1"
```

Listing 1. Simple PGA signature for TCP

```
ip1:
    sid: 1
    rev: 2
    args:
        - iphdr
    content: "iphdr->saddr == bpf_htonl(16843009)"
    comment: "SOURCE IP is 1.1.1.1"
```

Listing 2. PGA signature blacklisting a specific source IP address

The process of generating BPF code is simplified due to the architectural decision to represent signatures in C programming language code. An example of the conditional function generated based on a signature is presented in listing 3. The function is executed for every packet arriving TCP packet, and evaluates to "true" if given packet matches signature.

```
static __always_inline int tcp-sport_is_tcp-seq_0_1(
    struct tcphdr *tcphdr)
{
    /* some additional operations are possible */
    return tcphdr->source ==
           pga_2_byte32(tcphdr->seq, 0, 1);
}
```

Listing 3. PGA signature translated into code for BPF compiler

All helper BPF functions and macros are available for use in signature representation, e.g.:`bpf_htons`, `bpf_htonl`, `bpf_ntohs`, `bpf_ntohl` macros which translate short and long integers to network byte representation and back to host representation. Additionally, dedicated macros that allow for more comprehensive translation of signatures to C code are provided, i.e. `pga_1_byte16`, `pga_2_byte16`, `pga_1_byte32`, `pga_2_byte32` are used for extracting specific byte values from integers. Library of convenient helper functions and macros can easily be extended.

3.3 PGA Filter Integration in the GUARD Framework

PGA filter, as one of GUARD's Agents, is deployed inside use-case owner infrastructure, preferably at the edge of the local network. Agent is listening on the network interface at which all the incoming network traffic should be arriving. Each packet is analyzed and validated against the set of loaded DDoS signatures. If no signature is detected no actions are taken. However if packet matches one or more signatures, information about the possible malicious packet is recorded, i.e., source/destination IP address, source/destination port, timestamp, etc... and sent to the Kafka bus.

Data is placed in two channels: Results channel for GUARD Dashboard to visualize the information, and Network-Data channel that any GUARD Security

Service can subscribe to. Data is also dumped to Elasticsearch and can possible be used for detecting attack correlation and attack campaigns. Set of currently enabled signatures is a part of PGA filter configuration and can be modified on the go without letting a single packet through. Set of available PGA signatures is also a part of agent's configuration. Signature set is periodically updated by Security Controller service, which inquires NASK's database about newly generated PGA signatures. PGA signature database is semi-automatically (requires human input) updated with new signatures based on information coming from darknet traffic analysis (Fig. 8).

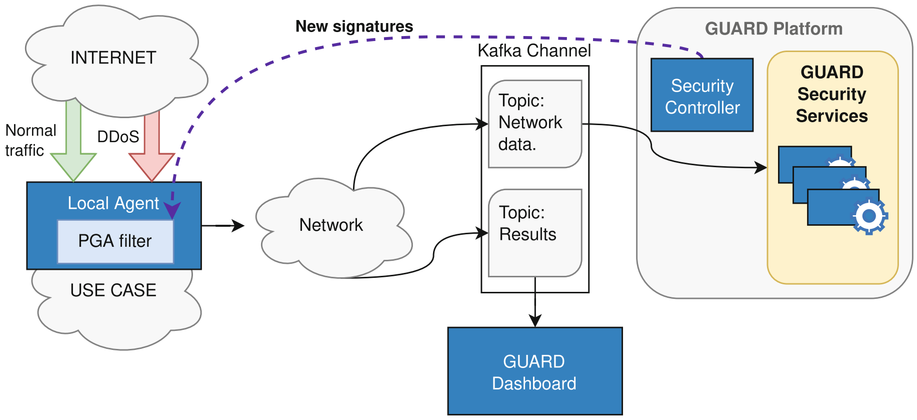

Fig. 8. Integration of PGA filter in the GUARD framework.

4 Conclusion

In conclusion, this work contributes to existing knowledge of denial-of-service prevention. It is our hope that presented approach will prove useful in developing new security solutions in the face of the growing threat posed by botnets. Although the concept of PGA signatures requires further development, this prototype shows the potential behind converting botnet fingerprints into rules for network traffic filtration. Based on the results achieved, levering the eBPF technology is a step in the right direction. Indeed, this solution provides a lightweight and portable approach, yet extensible enough for our use case while still allowing to operate on network packet at the lowest possible level in operating system. Improvements in the implementation of our module are however necessary to unlock the full potential of eBPF and XDP (cross-compiling, modularity). Future development should consider improving the overall experience and availability of our solution. Well-known, full-fledged IDS/IPS solutions like Snort[6] Suricata[7]

[6] https://www.snort.org/.
[7] https://suricata.io/.

should be taken as a point of reference to make our system more applicable in practice. Possibly, a way for integrating our concepts with above-mentioned platforms should be considered. Extending proved solutions seems like the most suitable way to integrate our approach in a common security ecosystem.

Acknowledgment. This work was supported in part by the European Commission under Grant Agreement no. 786922 (ASTRID) and no. 833456 (GUARD).

References

1. Antonakakis, M., et al.: Understanding the mirai botnet. In: Proceedings of the 26th USENIX Conference on Security Symposium. pp. 1093–1110. SEC2017, USENIX Association (2017)
2. Brownlee, N.: One-way traffic monitoring with `iatmon`. In: Taft, N., Ricciato, F. (eds.) PAM 2012. LNCS, vol. 7192, pp. 179–188. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28537-0_18
3. Cil, A.E., Yildiz, K., Buldu, A.: Detection of DDoS attacks with feed forward based deep neural network model. Exp. Syst. Appl. **169**, 114520 (2021). https://doi.org/10.1016/j.eswa.2020.114520, https://www.sciencedirect.com/science/article/pii/S0957417420311647
4. Deepa, V., Sudar, K.M., Deepalakshmi, P.: Detection of DDoS attack on SDN control plane using hybrid machine learning techniques. In: 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT). pp. 299–303 (2018). https://doi.org/10.1109/ICSSIT.2018.8748836
5. Dong, S., Abbas, K., Jain, R.: A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments. IEEE Access **7**, 80813–80828 (2019). https://doi.org/10.1109/ACCESS.2019.2922196
6. Hoque, N., Bhattacharyya, D.K., Kalita, J.K.: Botnet in DDoS attacks: trends and challenges. IEEE Commun. Surv. Tutor. **17**(4), 2242–2270 (2015). https://doi.org/10.1109/COMST.2015.2457491
7. Karpowicz, M.P.: Adaptive tuning of network traffic policing mechanisms for DDoS attack mitigation systems. Eur. J. Control **61**, 101–118 (2021). https://doi.org/10.1016/j.ejcon.2021.07.001, https://www.sciencedirect.com/science/article/pii/S0947358021000935
8. Kerrisk, M.: The UAPI header file split. https://lwn.net/Articles/507794/ (2012). Accessed 9 Feb 2022
9. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**, 80–84 (2017). https://doi.org/10.1109/MC.2017.201
10. Linux Kernel Community Contributors: L.T.: Linux kernel source code (2022). https://github.com/torvalds/linux. Accessed 9 Feb 2022
11. Liu, Y.: Improve DDoS botnet tracking with honeypots. https://www.botconf.eu/wp-content/uploads/2016/11/PR10-Improve-DDoS-Botnet-Tracking-With-Honeypots-LIU.pdf (2017). Accessed 9 Feb 2022
12. Makuvaza, A., Jat, D.S., Gamundani, A.M.: Deep neural network (DNN) solution for real-time detection of distributed denial of service (DDoS) attacks in software defined networks (SDNs). SN Comput. Sci. **2**(2), 1–10 (2021). https://doi.org/10.1007/s42979-021-00467-1
13. Malécot, E.L., Inoue, D.: The Carna botnet through the lens of a network telescope. In: FPS (2013)

14. Mallikarjunan, K.N., Muthupriya, K., Shalinie, S.M.: A survey of distributed denial of service attack. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1–6 (2016). https://doi.org/10.1109/ISCO.2016.7727096
15. Mansfield-Devine, S.: DDoS goes mainstream: how headline–grabbing attacks could make this threat an organisation's biggest nightmare. Netw. Secur. **2016**(11), 7–13 (2016). https://doi.org/10.1016/S1353-4858(16)30104-0, https://www.sciencedirect.com/science/article/pii/S1353485816301040
16. Moore, D., Shannon, C., Voelker, G., Savage, S.: Network telescopes: Technical report (2004–07)
17. Sharafaldin, I., Lashkari, A.H., Hakak, S., Ghorbani, A.A.: Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In: 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–8 (2019). https://doi.org/10.1109/CCST.2019.8888419
18. Silva, S.S., Silva, R.M., Pinto, R.C., Salles, R.M.: Botnets: a survey. botnet Activity: Analysis, Detection and Shutdown. Comput. Netw. **57**(2), 378–403 (2013). https://doi.org/10.1016/j.comnet.2012.07.021, https://www.sciencedirect.com/science/article/pii/S1389128612003568
19. SISSDEN Contributors NASK, USAAR, EXYS, DTAG, CYBE, MI: Deliverable d5.3: Final data analysis results (2019), https://sissden.eu/download/SISSDEN-D5.3-Final_Data_Analysis_Results.pdf Accessed 9 Feb 2022
20. Torabi, S., Bou-Harb, E., Assi, C., Karbab, E.B., Boukhtouta, A., Debbabi, M.: Inferring and investigating IoT-generated scanning campaigns targeting a large network telescope. IEEE Trans. Depend. Secur. Comput. **19**(1), 402–418 (2022). https://doi.org/10.1109/TDSC.2020.2979183
21. Zhang, X., Upton, O., Beebe, N.L., Choo, K.K.R.: IoT botnet forensics: a comprehensive digital forensic case study on Mirai botnet servers. Forensic Sci. Int.: Digit. Invest. **32**, 300926 (2020). https://doi.org/10.1016/j.fsidi.2020.300926, https://www.sciencedirect.com/science/article/pii/S2666281720300214