Code Augmentation for Detecting Covert Channels Targeting the IPv6 Flow Label

Luca Caviglione[1], Marco Zuppelli[1], Wojciech Mazurczyk[2,3], Andreas Schaffhauser[2], Matteo Repetto[1]
[1]Institute for Applied Mathematics and Information Technologies, Italy
{luca.caviglione, matteo.repetto, marco.zuppelli}@ge.imati.cnr.it
[2]FernUniversität in Hagen, Germany
{wojciech.mazurczyk, andreas.schaffhauser}@fernuni-hagen.de
[3]Warsaw University of Technology, Poland

2021 IEEE 7th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0522-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NetSoft51509.2021.9492661

***Abstract*—Information hiding is at the basis of a new-wave of malware able to elude common detection mechanisms or remain unnoticed for long periods. To this aim, a key approach exploits network covert channels, i.e., abusive communication paths nested within a legitimate traffic flow. The increasing diffusion of IPv6 makes it attractive for an attacker, especially for the presence of the Flow Label field, which can be manipulated to contain up to 20 secret bits per packet. Unfortunately, gathering data to implement a standalone detection mechanism or to support third-party security tools is a poorly generalizable process and often leads to scalability issues. This paper showcases how to take advantage of code augmentation features (i.e., the extended Berkeley Packet Filter) to detect covert channels targeting the IPv6 Flow Label. To prove its effectiveness, the proposed approach has been tested against Internet-wide traffic traces collected in the wild. Results indicate that it is possible to spot the channel while mitigating the memory footprint and the computational burden (e.g., the processed traffic only experience an additional delay of a few nanoseconds).**

***Index Terms*—code augmentation, stegomalware, network covert channels, IPv6, detection.**

I. Introduction

The increasing complexity of new malware as well as the proliferation of advanced attack schemes are severely challenging classical security appliances and mitigation methodologies, such as intrusion detection systems (IDS), firewalls and anomaly detection tools [1]. Among the various emerging threats, *stegomalware*, i.e., malicious software endowed with some form of steganography or information hiding techniques, is becoming a prime concern [2]. Stegomalware can exploit a vast array of techniques to improve its stealthiness or to bypass local security policies enforced via sandboxes. To this aim, a typical approach relies on the creation of a *covert channel*, i.e., a hidden communication path nested within a suitable carrier. Typical examples are processes or virtual machines colluding by exchanging information through the manipulation of shared hardware resources or software artifacts (see, e.g., [3] and [4]). From the perspective of an attacker, network traffic is very attractive: it can be used to remotely "transport" hidden information while offering a boundless and ubiquitous carrier [5]. *Network covert channels* can be used to support attacks in several manners: they can be used to exfiltrate stolen information, orchestrate nodes of a botnet or implement multi-stage loading architectures to extend malware functionalities at run-time [1]. Network covert channels can be created by directly hiding information in header fields, altering the structure of the packet, modulating the temporal evolution of the flow (e.g., throughput or jitter) or by exploiting complex interplays among multiple layers of the stack [6].

To gain advantage over the defendant, attackers are constantly searching for new carriers. The momentum gained by IPv6 is expected to lead to malware capable of establishing new hidden communication paths. Its feature-rich nature offers several opportunities for hiding data [7]. However, measurements in real-world scenarios limit the choice only to a few fields, as traffic characteristics and transitional mechanisms may impair the channel or reduce its stealthiness [8]. Among the others, methods targeting the `Flow Label` proven to be suitable for creating capacious and robust covert channels. Intended to help the network in routing operations, the `Flow Label` is seldom used in real-world deployments thus it can be manipulated for storing secret data [7], [8].

As a consequence, detection and mitigation of network covert channels are prime tasks to fully address the security of modern network scenarios. Unfortunately, literature mainly focused on threats exploiting IPv4, hence leaving the IPv6 counterpart largely unexplored [6], [9]–[11]. Besides, the inspection process is often tightly coupled with the used information hiding method: this makes the detection poorly generalizable and could lead to non-negligible computational burdens (e.g., to check protocol fields via deep packet inspection [12]).

A promising approach for counteracting stegomalware leverages code augmentation features offered by the Linux kernel [13]. Specifically, the extended Berkeley Packet Filter (eBPF) appears to be suitable for gathering at run-time information on different protocol behaviors to enable the detection of covert communications [14], [15]. Therefore, in this paper we present a method that uses the eBPF to monitor the usage of the IPv6 `Flow Label` in a computational-efficient manner. Such data can be combined with information provided by other monitoring or security tools deployed in the network (e.g., firewalls and IDS), which are insensitive to covert channels targeting IPv6 [8]. According to [16], the algorithm implemented in many operating systems for generating the

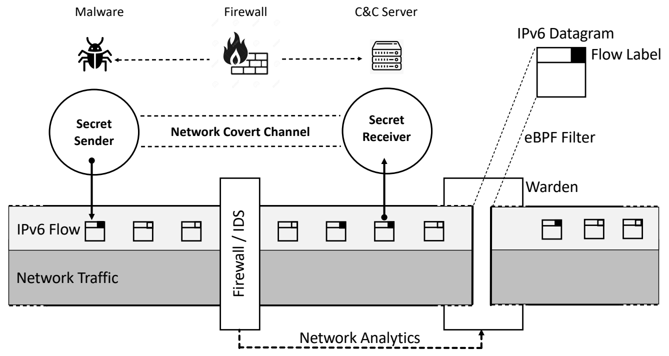


Fig. 1. Attack model and reference scenario considered in this work. Black squares denote a `Flow Label` altered to contain secret information.

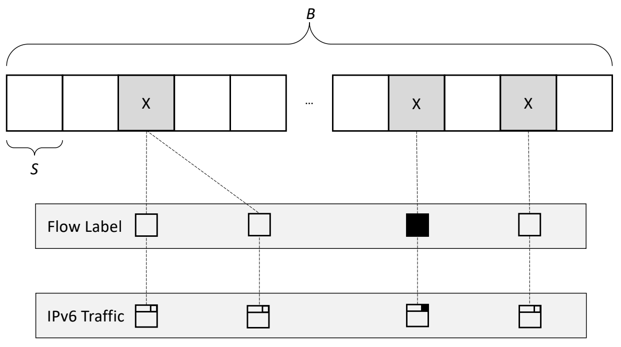


Fig. 2. Mapping of the IPv6 `Flow Label` field in the bin space. Black squares denote a `Flow Label` altered to contain secret information.

`Flow Label` could be flawed, thus the inspection of its behavior can further improve the overall security.

The contribution of this work is twofold. First, it proposes a lightweight method for spotting the presence of a covert communication hidden in the `Flow Label` field, also by taking into account measurements provided by other network security and monitoring tools. This is of prime importance, since modern intrusion detection systems are affected by major drawbacks when handling IPv6 traffic [17] and mostly fail to detect covert channels out of the box. Second, it contains a thorough set of tests performed with real traffic traces, whereas many other works only focus on synthetic or simulated data.

The rest of the paper is structured as follows. Section II introduces the attack model and the reference scenario, while Section III discusses the data gathering strategy implemented via code augmentation and the related detection methodology. Section IV showcases numerical results. Finally, Section V concludes the paper.

II. Attack Model and Reference Scenario

The attack model considers two secret endpoints that want to remotely communicate. To this aim, they create a covert channel by eavesdropping an existing overt IPv6 flow from the bulk of network traffic, and then inject secret information within the `Flow Label`. To improve both confidentiality and security of the hidden communication, secrets can be encrypted. This threat model is usually exploited by an attacker who wants to exchange data from the host/device of the victim towards a remote Command & Control (C&C) server while avoiding detection or blockages. As a possible example, the attacker can inoculate a malware via a phishing campaign and then exfiltrate sensitive data or retrieve an additional payload [1], [2]. Fig. 1 depicts the general attack model.

Concerning the reference scenario, we assume that a *warden*, i.e., a node able to inspect the traffic containing the covert channel, is present somewhere in the network. For the sake of simplicity, Fig. 1 portraits a warden deployed at the end of the covert channel. Typically, the placement of the warden can be arbitrary, except when the attacker uses some form of reversibility for restoring datagrams to their original form [18]. Since reversible channels have not been observed in the wild, this work will not consider such a class of threats. To spot the channel, we consider a warden that inspects all packets of the incoming network stream through an eBPF program and creates statistics about the usage of Flow Label values. The warden also teams up with a network analysis or security tool (e.g., an IDS), which provides standard measures as well as descriptive behaviors of the underlying deployments, for instance in terms of traffic volumes, Netflow statistics, or feedbacks provided by network intrusion detection tools.

III. Packet Inspection and Detection Methodology

In this section, we describe how information on the `Flow Label` is collected and organized. Then, we showcase the proposed detection methodology.

A. Packet Inspection and Organization

For the case of detecting covert channels targeting the `Flow Label`, we developed a specific tool that creates statistics about the usage of such values[1]. As hinted earlier, packet inspection is implemented through eBPF. In essence, this framework allows to extend the kernel to count the occurrence of `Flow Label` values. To this aim, we set a hook point in the `tc` queue management, which enables to run an eBPF program for each IPv6 packet to be processed. To provide scalability properties and prevent performance degradation in presence of large volumes of traffic, the 20-bit space of possible values is mapped into a bin-based data structure composed of B equally-capable bins, as depicted in Fig. 2. The mapping is based on the first $\log_2 B$ bits of the `Flow Label`, which are used to index the array of bins. With S we denote the size in bits of each bin: in our case, $S = 2^{20}/B$ bits. Such a mechanism demonstrated to be lightweight as the traffic processed by our framework experiences only delays of $\sim$10 ns per packet.

The bin data structure is implemented via a key-value map and located in a shared memory area. Data is then periodically collected by a user-space utility (see [13], [15] for more details). We denote with Δt the time in seconds between two

[1]`bccstego` tool available online: https://github.com/mattereppe/bccstego.

consecutive readings of such a map. The same utility computes the number N of non-empty bins, i.e., "dirty" bins that have been flagged by the inspection code.

As it will be discussed in Section IV, N provides an estimate of the current number of flows. This is only an approximation, because different `Flow Label` values share the same bin, thus causing collisions. Greater values of B reduce such a probability and improve the precision, but at the price of a higher memory burden. In addition, the estimation is subject to "saturate" to B, while the bins are gradually flagged. To avoid such an effect, bins are periodically emptied and the value of N is accordingly restored to 0 by programmatically scanning the key-value structure. As it will be detailed in the following, a suitable tradeoff between accuracy and performances should be searched for. Fig. 2 depicts the resulting data structure.

As an example, let us consider the case of $B = 2^{12}$ bins with a size $S = 2^8$ values. If a packet with a `Flow Label` value equal to 337 (i.e., `0x00151`) is observed, the second bin is flagged since it is the one containing values in the $256-511$ range (indexed by the `0x001` prefix). Accordingly, N is incremented by 1.

B. Detection Methodology

Our detection methodology is based on the coarse grained estimation N of the number of flows, computed from the statistics gathered in the bin-based data structure. To avoid burdening the notation, we will drop any dependence on time t, except when doubts arise.

Since each IPv6 conversation is identified via a fixed, unique `Flow Label` value generated according to a uniform distribution [19], N can provide a coarse estimate of the number of flows. Recalling that measurements are retrieved from eBPF every Δt and bins are periodically emptied to avoid saturation, we considered two different windowing mechanisms, based on a fixed temporal duration or a fixed number of samples (i.e., of readings of the bin-based map), denoted with T and W, respectively. As a consequence of this "grouping" scheme, the meaningful values of N are those available at the end of each window, i.e., just before values are reset. In general, other values could be discarded or stored to compute suitable models or datasets to be processed with machine-learning-based frameworks. Nevertheless, the ability of collecting multiple measurements within a given window guarantees the ability of adjusting the proposed approach to match measurements/feedback information provided by external tools with different timings. The presence of an anomalous behavior in the usage of the `Flow Label` can be then detected by observing deviations of N from an expected value, for instance by feeding a model based on past observations, or by comparison with network analytics reports provided by a complementary tool (see Fig. 1 as a paradigmatic example). Without loss of generality, we assume to have periodical measurements on the number of active IPv6 flows in the network, denoted as F, which is a metric commonly available from several security appliances.

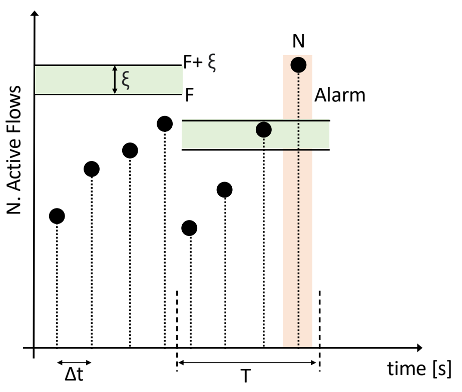


Fig. 3. Detection strategy based on the bin-based `Flow Label` partitioning.

Fig. 3 depicts an example when measurements on non-empty bins are grouped in a window of T seconds. As shown, the basic idea is that if the number of flagged bins (e.g., those for which the corresponding `Flow Label` values have been observed in the considered window) is greater than the number of observed active IPv6 flows, a covert communication could be present and an alarm is raised. To adjust the responsiveness of the approach (e.g., to limit false positives), with ξ we denote a guard threshold that has to be exceeded.

Unfortunately, a direct comparison between N and F (including the guard ξ) is seldom possible in real deployments. In fact, huge traffic volumes could account for several concurrent flows (i.e., high values of F), thus requiring a fine-grained bin partitioning. Indeed, large bins (i.e., small B) may cause masking phenomena, as a single bin can represent up to S different flows/values. Therefore, we introduce a scale factor for adjusting the impact of the imperfect partitioning caused by $B < 20$, collisions of real or artificial `Flow Label` values populating the same bin, and non-optimal matching between the traffic volume and the parameters ruling the detection, i.e., W, T, and Δt. The channel is considered present when the following equation holds:

$$\alpha N > F + \xi \tag{1}$$

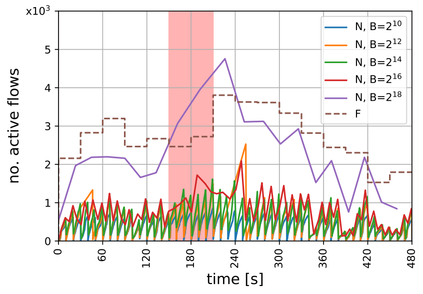
where, α is the scale factor to balance the flow/bin proportion. We point out that, for the sake of brevity, in the following, we will not consider the impact of ξ, which is left as a future development.

IV. Experimental Results

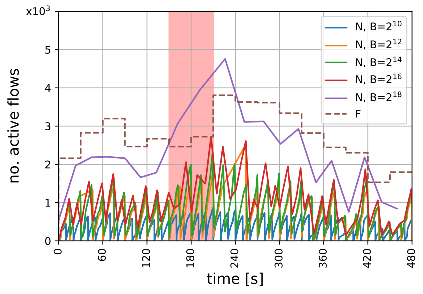
To quantify the effectiveness of the proposed approach for detecting covert channels, we carried out a thorough performance evaluation campaign. In the following, we first review the experimental testbed, then we discuss numerical results.

A. Testbed Preparation

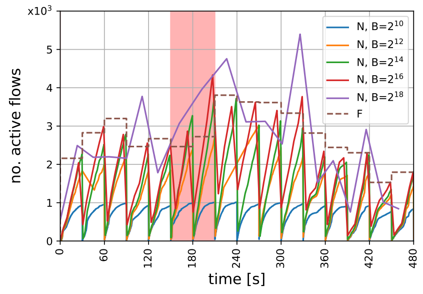
To conduct trials, we considered two secret endpoints exchanging data through a covert channel targeting the `Flow Label` field of IPv6. The communicating peers have



(a) $T = 10$

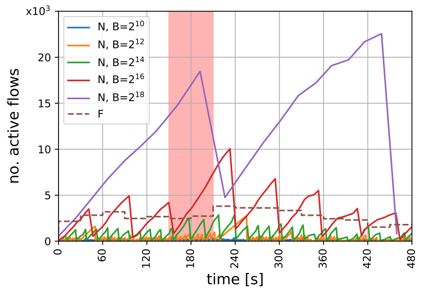


(b) $T = 15$

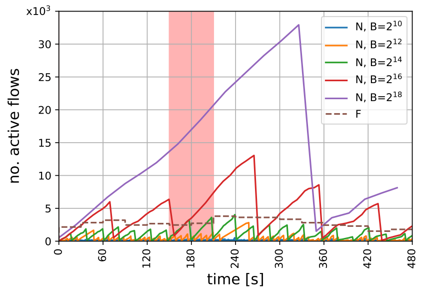


(c) $T = 30$

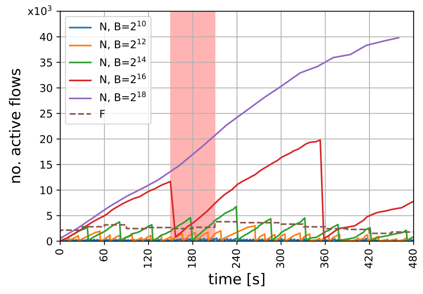
Fig. 4. Number of changing bins of the observed traffic for different values of T. The red area denotes when a covert communication is present.



(a) $W = 10$



(b) $W = 15$



(c) $W = 30$

Fig. 5. Number of changing bins of the observed traffic for different values of W. The red area denotes when a covert communication is present.

been implemented via two virtual machines running Debian GNU/Linux 10 (kernel 4.20.9), with 1 virtual core and 4 GB of RAM. A third virtual machine with the same characteristics has been deployed to route the traffic, run eBPF scripts and the user-space daemon written in Python. To run the virtual machines, a host with a 3.60 GHz Intel i9-9900KF CPU, 32 GB of RAM and Ubuntu 20.4 (Linux kernel 5.8.0) has been used. Measurements (i.e., the values of N) are sampled every $\Delta t = 100$ ms. We point out that, in realistic network conditions, sampling intervals could be less tight: this value has been selected to have a worst-case in terms of computational burden. To implement the covert channel, we created an ad-hoc Python module using Scapy 2.4.3 for manipulating packets and NetfilterQueue 0.8.1 for intercepting the network traffic.

To test our idea in realistic network conditions, we replicated legitimate background conversations from traffic collected on a OC192 link between Sao Paulo and New York on January 17, 2019 from 14:00 to 15:00 CET and made available by the Center for Applied Internet Data Analysis[2]. Without loss of generality and to prevent burdening our trials, we removed packets with a `Flow Label` value equal to 0, ICMPv6 traffic and single-datagram UDP conversations. The resulting dataset was solely composed of IPv6 traffic with $\sim 3,000$ active connections, on average. Traces have then been processed via `tcprewrite` to rebuild the payload and properly assign MAC values. The resulting traffic has been transmitted with `tcpreplay`. The overt IPv6 flow used by the secret endpoints to implement the covert channel has been produced via an `scp` transfer with a length adequate for containing secret messages of 65, 130, and 327 kbit. To prevent having a favorable setting, the content of messages have been randomly generated. This models an attacker who adopts an additional encryption layer or scrambling scheme to mimic the random nature of legitimate `Flow Label` values, in order to avoid statistical signatures that can make the detection easier [8]–[11].

To have a fair comparison, the background traffic has been limited to 2 Mbit/s, whereas the overt traffic to 500 kbit/s. Such values allowed to model a stegomalware acting into a medium-sized deployment trying to exfiltrate some sensitive data, retrieve a payload containing many sophisticated attack routines, or exchange commands multiplexed in a unique hidden stream with a C&C server to orchestrate a botnet.

As discussed, our method can be jointly used with other network monitoring tools. In our trials, we used `nProbePro`[3] to inspect the traffic in real-time and compute the number of active IPv6 flows F. According to additional tests, the number of active flows reported by `nProbePro` is insensitive to the presence of IPv6 covert channels. This further supports the need of teaming up with a specific solution when IPv6 `Flow Label` channels have to be detected. In our testbed, a new

[2]The CAIDA Anonymized Internet Traces Dataset (April 2008 - January 2019) - Used traces: Jan. 17th 2019. Available online: https://www.caida.org/data/monitors/passive-equinix-nyc.xml [Last Accessed: Feb. 2021].

[3]`nProbePro` v. 9.5.210129, https://www.ntop.org/products/netflow/nprobe/ [Last Accessed: Feb. 2021].

measure of F is provided by `nProbePro` every 30 s, thus we assume such a value as the best granularity for network analytics information.

B. Sensitivity Analysis

The first round of tests aimed at understanding the behavior of the proposed framework when changing different parameters ruling the detection, namely the number of bins B and the window size T and W. To this aim, we considered a mapping space composed of $B = 2^{10}, 2^{12}, 2^{14}, 2^{16}, \text{and } 2^{18}$ bins. For what concerns how to group measurements and reset the number of active flows, we considered windows of different durations, i.e., $T = 10, 15, \text{and } 30$ s, and of different sizes, i.e., $W = 10, 15, \text{and } 30$ samples. This round of tests lasted 8 minutes and considered a covert transmission of 65 kbit. Fig. 4 and Fig. 5 showcase the obtained results. In all trials, the scale factor α has been set equal to 1.

First, we elaborate on the impact of the number of bins B over the accuracy of the estimation. In general, higher values of B lead to a number of non-empty bins that efficiently reflect the evolution of active IPv6 flows. In other words, the trend of N is close to the one of F as B increases. However, when the number of bins is smaller than the average number of flows composing the traffic, the bulk of `Flow Label` quickly saturates the few bins available: this effect is largely predominant in case of longer windows, e.g., see the case $B = 2^{10}$ for $T = 30$ s in Fig. 4(c).

As a possible workaround, one might reduce the length of the window (see Fig.s 4(a) and 4(b)) but at the price of discarding measurements too often. Consequently, N is not able to "follow" with a sufficient precision the trend of measured active IPv6 flows F provided by `nProbePro`. As it will be discussed, this mismatch can be partially mitigated via the α scale parameter.

Fig. 5 investigates the same issues when a fixed number of samples is collected (i.e., W increases). In general, similar considerations can be drawn as in the previous case. For instance, the longer the window used to spot the presence of the channel, the better the ability to estimate network conditions. However, the "saturation" effect of using an insufficient number of bins is less visible in this case, because of the scale. Instead, it is possible to observe that the "sawtooth" behavior becomes less evident and in some scenarios (e.g., $B = 2^{18}$ and $W = 30$ depicted in Fig. 5(c)) the evolution of N exhibits an apparent unbounded growth. This can be ascribed to performance issues. Specifically, even if the desired sampling time is set to $\Delta t = 100$ ms, it turned out that the user-space daemon was not able to dump measurements collected by the eBPF with such a frequency, mainly due to the delay introduced by the huge number of memory operations (especially for $B = 2^{16}$ and $B = 2^{18}$). As a consequence, samples for N are produced with a slow dynamic and the time to collect W values increases, and the corresponding "resetting interval" is postponed. Therefore, the number of flows actually captured in the bin-based structure is greater than the one of the corresponding time-limited window. Fig. 5 clearly shows, for the case of $B = 2^{18}$, a trend of N that overestimates the number of flows. This is due to the time needed to read the bin structure, which in turn extends the windows with the side effect of considering more flows. Instead, the desired sampling time has been guaranteed only with the lower number of bins: this leads to the more regular and smooth evolution of N.

C. Covert Channel Detection

The second round of tests aimed at quantifying the effectiveness of the bin-mapping scheme to spot the presence of IPv6 covert channels. In this case, we consider two different hidden communication attempts: a single exfiltration of 130 kbit of information (e.g., a document), and two communications of 65 kbit and 327 kbit, respectively (e.g., exchange of parameters for configuring a backdoor and the download of a remote malicious payload). Attacks will be performed in a timeframe of 15 minutes. For the sake of compactness, as discussed in Section IV-B, we limit our investigation to $B = 2^{14}, 2^{15}, \text{and } 2^{16}$ and $T = 30$ seconds as to have a one-to-one temporal matching with the value F provided by `nProbePro`.

Concerning the single-channel case, Fig. 6 depicts the outcomes of the detection when the rule defined in Eq. (1) is used, with $\alpha = 1$. As shown, higher values of B tends to generate many false positives, as the number of bins overestimates the amount of active IPv6 conversations. Yet, when the covert communication is present, the higher number of `Flow Label` values generated by the injection of the secret data is correctly reflected into greater values of N. Similar behaviors can be observed when in the presence of two different covert channels as shown in Fig. 7. Specifically, $B = 2^{16}$ leads to a larger number of false positives, which can be mitigated by a suitable choice of ξ. However, defining such a parameter a priori is a difficult task as it depends on the scenario (e.g., the characteristics of the background traffic) and the design choices of the overall detector, such as Δt, T and B. This investigation is part of our ongoing research. Instead, $B = 2^{15}$ naturally offers the best matching between the volume of conversations composing the IPv6 traffic and the number of bins, thus leading to best performances in terms of steadiness of the detection.

Fig. 8 showcases the impact of the parameter α on the performance in terms of detection of the covert channel. In this case, by using $T = 15$ s, it is possible to capture the evolution of the traffic with a more fine-grained dynamic compared to the case of $T = 30$ s. However, smaller timeframes account for a more frequent "reset" of the bin-based scheme and N tends to underestimate the amount of active IPv6 conversations. Thus, it is not possible to directly compare N with the measurement F provided by `nProbePro`. To this aim, α can correct this mismatch by magnifying the obtained values. Unfortunately, this also amplifies possible measurement errors and leads to further false positives. As shown, with $\alpha = 1.4$, an adequate tradeoff in terms of detection (i.e., the channel is correctly spotted for almost its entirety) and false positive is found.

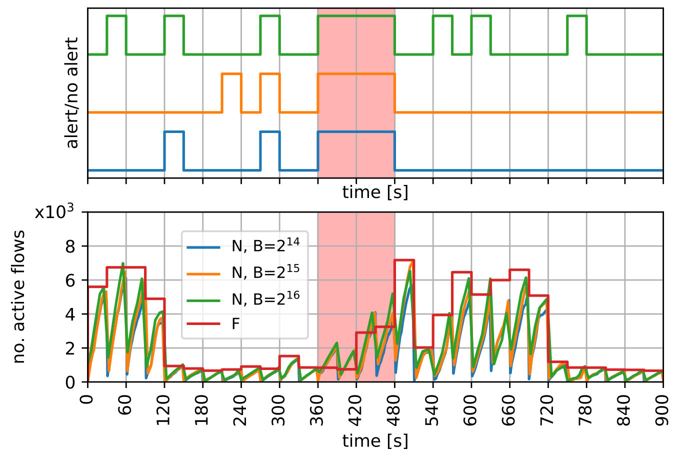


Fig. 6. Detection of a covert channel transmitting 130 kbit of secret data. The red area denotes when a covert communication is present.

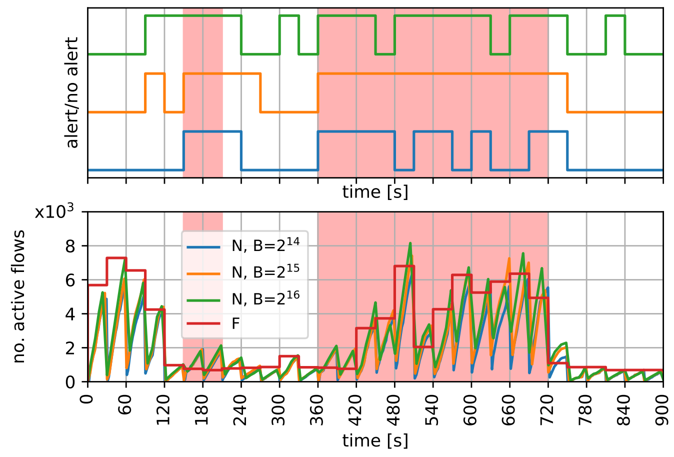


Fig. 7. Detection of two covert channels transmitting 65 and 327 kbit of secret data, respectively. The red areas denote when a covert communication is present.

D. Resource Utilization and Scalability

We also investigated the performances of the proposed code augmentation approach, by evaluating both the impact of the eBPF filter and the user-space program. In particular, we measured the CPU usage and the memory consumption with different values of B. It turned out that, with lower values of B, the amount of resources required by the user-space program is minimal. Specifically, with $B = 2^{10}$, the used CPU is approximately 7.3%, whereas with $B = 2^{18}$, it rises up to 20%.

For higher values of B, the user-space program is not able to sustain the chosen sampling interval, i.e., $\Delta t = 100$ ms. For low values of B (i.e., $B = 2^{10}$), the actual sampling time was equal to 154 ms, on average. Instead, with $B = 2^{18}$, the user-space program was unable to provide measurements in a timely manner, thus inflating the Δt to ~26 seconds, on average. This can be also observed in Fig. 4 and Fig. 5:

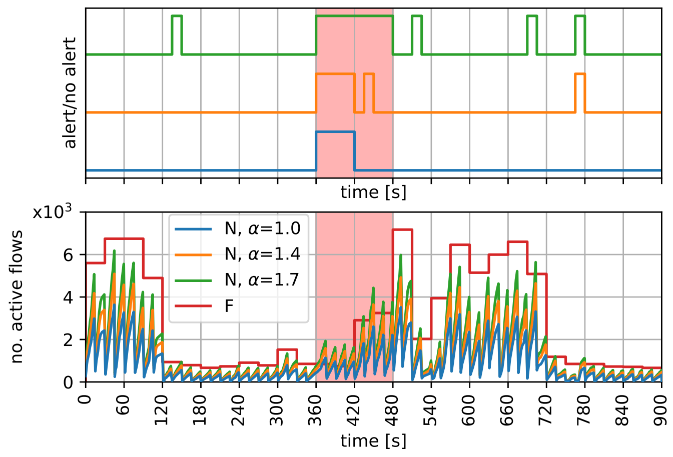


Fig. 8. Detection of a covert channel transmitting 130 kbit of secret data. Size of window $T = 15$ s. The red area denotes when a covert communication is present.

the lack of a precise timing accounts for temporal evolutions of N with less/more samples. However, further tests have demonstrated that such a fine granularity is not strictly required to improve the accuracy of the detection, because good results can be achieved with sampling times of a few seconds. Unfortunately, the eBPF framework was originally designed as a low-overhead kernel monitoring tool, and there are several limitations in terms of available features and programming paradigms (e.g., LRU hash tables and batch reading) [20].

E. Discussion and Limits

As shown, code augmentation is suitable for developing simple filters able to inspect network traffic and detect covert channels. In general, adding a filter requires to write small portions of code and multiple filters can be deployed to concurrently gather data from different portions of the traffic, which is vital when the target of the covert communication is unknown. This paradigm can also help to "enrich" information provided by an IDS, firewalls and network monitoring tools insensitive to information-hiding-capable attacks or not ready to fully handle IPv6 security. Unfortunately, there is not a one-fits-all solution and each filter-detection pair requires tuning. For instance, the `Flow Label` case requires to tune the bin-based structure for having a suitable matching between the observed traffic and the detection strategy. Therefore, the proposed mechanism should be properly engineered according to the scenario that has to be protected. As an example, the average traffic volume and number of hosts should be known to select the granularity of bins (i.e., B and S) and avoid under-/overestimations. Similar considerations are valid for the parameters α and ξ.

Even if performances of the eBPF were satisfactory in terms of memory occupation and CPU utilization when handling Internet-wide traffic, we encountered some issues. First, we were unable to run filters with a number of bins $B > 2^{18}$. This limitation could be ascribed to security reasons enforcing constraints in the amount of memory that can be used in the kernel. A possible workaround would be to use a different data structure for the bins. Currently, we use a vector map, which is statically allocated to hold all B bins, but a dynamic map would reduce memory usage though it requires a memory management technique to periodically remove older entries. In this extent, the adoption of a hash-based approach to improve performance is part of our future developments. Another constraint we experienced concerns the user-space daemon, which must scan the whole bin map for estimating N. With high values of B, we were unable to sustain refresh rates of $\Delta t = 100$ ms. Even if production-quality tools for network security and analysis seldom need such a granularity (e.g., to ensure scalability and statistical relevance), removing this limitation can be useful for deploying the proposed framework in resource-constrained network appliance.

Lastly, an additional limitation of the approach concerns how the channel is spotted. In fact, the use of a windowing scheme shifts the detection of T seconds or upon completely receiving W samples. This may lead to the impossibility of blocking shorter covert communication attempts in real time, but only to raise an alarm *a posteriori*.

V. Conclusions and Future Works

In this paper, we have presented a code augmentation mechanism for the detection of covert channels targeting the IPv6 `Flow Label`. To this aim, we developed an eBPF filter for mapping observed values into a bin-based structure and roughly estimate the volume of conversations. Covert communications can then be revealed via discrepancies between the observed data and measures provided by a network analysis tool. Results prove the feasibility of using code augmentation to support other tools unable to deal with network covert channels out of the box.

Future work is primarily devoted to conduct an extensive evaluation campaign, especially to quantify the proposed detection framework in terms of false positives and false negatives as well as the impact of ξ. Owing to the flexibility of the eBPF framework, part of our ongoing research aims at using the proposed approach with other types of channels. For instance, eBPF can be used to inspect the `Traffic Class` and `Hop Limit` fields within the IPv6 header, which allows an attacker to store or modulate secret information, respectively. Another future development concerns the utilization of eBPF for actively manipulating traffic, e.g., to sanitize flows and disrupt the channels by overwriting fields or restoring them to a standard value.

Acknowledgment

This work has been supported by the EU Project ASTRID, Grant Agreement No 786922, and by the EU Project SIMARGL, Grant Agreement No 833042.

The authors would like to thank ntop (www.ntop.org) for providing a free license of `nProbePro` used in experiments.

References

[1] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, 2020.

[2] W. Mazurczyk and L. Caviglione, "Information hiding as a challenge for malware detection," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89–93, 2015.

[3] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the communication between colluding applications on modern smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 51–60.

[4] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "Containerleaks: Emerging security threats of information leakages in container clouds," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 237–248.

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

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

[7] N. Lucena, G. Lewandowski, and S. Chapin, "Covert channels in IPv6," in *Int. Workshop on Privacy Enhancing Technologies*. Springer, 2005, pp. 147–166.

[8] W. Mazurczyk, K. Powójski, and L. Caviglione, "IPv6 covert channels in the wild," in *Proceedings of the 3rd Central European Cybersecurity Conference*, 2019, pp. 1–6.

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

[10] W. Mazurczyk and L. Caviglione, "Steganography in modern smartphones and mitigation techniques," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 334–357, 2014.

[11] L. Caviglione, "Trends and challenges in network covert channels countermeasures," *Applied Sciences*, vol. 11, no. 4, 2021.

[12] A. Salih, X. Ma, and E. Peytchev, "Implementation of hybrid artificial intelligence technique to detect covert channels attack in new generation internet protocol IPv6," in *Leadership, Innovation and Entrepreneurship as Driving Forces of the Global Economy*. Springer, 2017, pp. 173–190.

[13] A. Carrega, L. Caviglione, M. Repetto, and M. Zuppelli, "Programmable data gathering for detecting stegomalware," in *Proceedings of the 2nd International Workshop on Cyber-Security Threats, Trust and Privacy Management in Software-defined and Virtualized Infrastructures (SecSoft)*. IEEE, 2020.

[14] L. Deri, S. Sabella, and S. Mainardi, "Combining system visibility and security using eBPF," in *Proceedings of the Third Italian Conference on Cyber Security, Pisa, Italy, February 13-15, 2019*, ser. CEUR Workshop Proceedings, P. Degano and R. Zunino, Eds., vol. 2315. CEUR-WS.org, 2019.

[15] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli, "Kernel-level tracing for detecting stegomalware and covert channels in linux environments," *Computer Networks*, p. 108010, 2021.

[16] J. Berger, A. Klein, and B. Pinkas, "Flaw label: Exploiting IPv6 flow label," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 1594–1611.

[17] B. Blumbergs, M. Pihelgas, M. Kont, O. Maennel, and R. Vaarandi, "Creating and detecting IPv6 transition mechanism-based information exfiltration covert channels," in *Nordic Conference on Secure IT Systems*. Springer, 2016, pp. 85–100.

[18] W. Mazurczyk, P. Szary, S. Wendzel, and L. Caviglione, "Towards reversible storage network covert channels," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–8.

[19] S. Amante, B. Carpenter, S. Jiang, and J. Rajahalme, "IPv6 flow label specification," Internet Requests for Comments, RFC Editor, RFC 6437, November 2011.

[20] S. Miano, M. Bertrone, F. Risso, and M. Tumolo, "Creating complex network service with ebpf: Experience and lessons learned," in *Proc. IEEE High Perform. Switching Routing (HPSR)*, Bucharest, Romania, Jun., 18th-20th 2018, pp. 1–8.