Demystifying the Role of Public Intrusion Datasets: a Replication Study of DoS Network Traffic Data

Marta Catillo[a,*], Antonio Pecchia[a], Massimiliano Rak[b], Umberto Villano[a]

[a] *Università degli Studi del Sannio, Benevento, Italy*
[b] *Università degli Studi della Campania Luigi Vanvitelli, Aversa, Italy*

Abstract

Public intrusion datasets are contributing to make security research accessible to a large community of users, but are often trusted and reused neglecting the actual impact of the attacks therein on victim services. This paper documents a study aimed to assess whether the attacks provided by public datasets are impactful on their targets. DoS traffic data from five public datasets (CICIDS2017, ISCXIDS2012, NDSec-1 2016, MILCOM 2016 and SUEE 2017) are replayed, monitoring the performance of the victim server under different defense, configuration and load conditions. The obtained results show a partial ineffectiveness of the attacks of the datasets in the presence of defense mechanisms and suitable server configurations. These results pave the way for the construction of more rigorous datasets, collected on documented and realistic server configurations and reflecting actual traffic conditions under normative operations and disruptive attacks.

Keywords: Denial of Service, traffic replay, web server, availability

1. Introduction

As the risk of cyber attacks constantly grows through the years, the use of data collected in normal and altered system state is widely recognized as a mean to discriminate between normal operating conditions and anomalous ones. **Security data** can be collected from multiple sources that range from customary *system* and *application logs* to specialized tools such as *intrusion detection systems*, *network audit agents*, *integrity monitors*. Insightful inspection of data can help system administrators to develop situational awareness, to detect and classify security incidents, and to set up countermeasures and

*Corresponding author (phone: +39-0824-305805)

Email addresses: `marta.catillo@unisannio.it` (Marta Catillo), `antonio.pecchia@unisannio.it` (Antonio Pecchia), `massimiliano.rak@unicampania.it` (Massimiliano Rak), `villano@unisannio.it` (Umberto Villano)

defense strategies [1], [2], [3], [4], [5]. Unfortunately, due to obvious confidentiality and non-disclosure reasons, publicly-available data relative to *spontaneous* (i.e., neither induced nor simulated) security attacks and incidents are lacking. In consequence, research on network, system and application security is often carried out by experimentation on data collected through honeypots and "lab-made" intrusions. In particular, research on intrusion detection systems (IDS) customarily relies on **public datasets**, collected by network tools in a controlled testing environment trying to mimic realistic attack conditions. Available intrusion datasets are surveyed in [6] and include the widely used KDD-CUP'99, UNSW-NB15, NDSec-1 2016, CICIDS collections, to mention some examples.

Public datasets have become common benchmarks for intrusion detection techniques and tools. As a matter of fact, security researchers and practitioners use the detection figures obtained on known datasets as yardstick to measure the validity of their proposals. For example, recent trends put forth a massive –and ever-increasing– body of papers on machine-learning-based intrusion detectors, which are typically assessed by attempting to detect attacks from public datasets with high recall, precision and accuracy. Surprisingly, in spite of the consolidated usage of public datasets, the problem of their **representativeness**, linked to often-lacking information on collection testbeds and modalities, configuration of nodes and applications, workloads, attack types and impact, has been hardly ever dealt with in the literature on security research. A notable example is the work on the KDD-CUP'99 dataset. Authors in [7] indicate that KDD-CUP'99 lacks up-to-date attack classes and contains duplicate records. These same considerations hold for the *younger* NSL-KDD dataset, which is intended to solve some of the inherent problems of KDD-CUP'99; however, it still suffers from several limitations originally discussed in [8]. Even taking for granted the validity of the collection modalities of more recent datasets, there is a further issue to consider. The doubt is whether the attacks used for generating these datasets are actually effective against their targets, or are a just a sort of "temporary disturbance" that can be tolerated, possibly with no effect, by present-day hardware and software systems. In the second case, we would paradoxically find that a substantial body of work was produced by researchers training new detection systems and assessing their validity on the top of attacks that have no actual harmful effect on their targets.

This paper develops around the observation that public datasets are contributing to make security research accessible to a tremendous community of users; however, we observe that datasets are *blindly* trusted and reused neglecting the **actual impact** of the attacks therein on availability and performance of operations of the victim services. Differently from this common practice, our study aims to assess whether the attacks provided by public datasets are impactful, and to shed some clear light on the factors that determine their effectiveness. This work puts forth an unprecedented perspective on public datasets: to the best of our knowledge, we are aware of no similar studies. Our proposition is explored in the context of publicly-available traffic data gathered under **Denial of Service** (DoS) attacks [9]. The class of DoS attacks is typically available in any public dataset and keeps attracting efforts by many research groups. For example, countermeasures have been developed to mitigate DoS attacks [10]; more important, a plethora of DoS detectors have spread in the literature given the rapid growth of deep learning methods and tools [11]. DoS detection is the typical use case of public datasets where researches tend to dig into machine learning facets –detection rate is now close to 100% in many cases– rather than reasoning on the representativeness of the datasets and how it may bias detection results in practice.

In a previously-appeared paper, we documented a preliminary experiment with *one attack* from the widely used CICIDS2017 dataset [12]. Here we present a much wider study with CICIDS2017, ISCXIDS2012, NDSec-1 2016, MILCOM 2016 and SUEE 2017, a large mixture of different DoS attacks and a detailed investigation of the impact of defense, which contribute to comprehensive experiments and findings along different directions. For all the datasets mentioned above, collected and made available by independent research teams over the past years, traffic data are provided in the form of `pcap` **packet data files**, which are typically produced by many common network capture utility programs. In order to understand the hype around these datasets, it is interesting to note that, after three years since its publication, the reference CICIDS2017 paper [13] is rapidly approaching 700+ Google scholar citations at the time of this writing. Our evaluation approach *in a nutshell* consists in **replaying** attack traffic data stored in public `pcap` files against a victim web server in a controlled testbed; during the replay of the attack, the victim is continuously monitored to collect the typical metrics of *throughput*, *reply time* and *throughput loss*. To conduct the experiments we propose a support tool called RELIVE, which allows to "relive" previously-captured traffic data over brand new sockets and connections. The campaign consists in repeated experiments where each attack is replayed under different combinations of key factors, i.e., presence of *defense*, *configuration* of the victim server and *load conditions*. It is worth noting that all of these can alter the impact of a given DoS attack; however, assessed datasets are not accompanied by specific details covering any of them. Our experiments indicate that only few attacks are effective in case of realistic operating conditions. The key outcomes and findings of our study –with respect to the datasets and system in hand– are:

- *DoS traffic provided by public datasets suffers from the presence of proper defense mechanisms.* Most of the public attacks assessed in the paper are strongly mitigated by a defense module included by the default installation of the victim server. Surprisingly, we had to "manually" disable some deference mechanisms –thus render the victim much less secure than expected– in order to make the attacks effective. In consequence, most of the attacks would be negligible against real-life servers where proper defense is reasonably in place.

- *The configuration of the victim server has a major impact on the effects of a DoS attack.* We observe that the effects of many public attacks assessed here disappear against a "tuned-up" configuration of the web server, i.e., by boosting its capacity and multithreading capability. For example, in our study the loss of throughput of the victim under several attacks drops from 100.0%, i.e., service unavailable, to 0%, i.e., no impact, after adjusting the server configuration.

- *A DoS attack per se explains only a portion of the performance loss of the victim server.* "Benign" load served by the victim (i.e., load generated by legitimate clients) must be properly accounted when characterizing a DoS. As for the public attacks, we did not observe an advantageous interplay between attack traffic and increasing load. The sharp separation of malicious/benign traffic –typical of many public datasets– does not properly reflect uncertainty of real-life operations.

The rest of this paper is organized as follows. Section 2 presents related work on public datasets, pertinent literature on DoS attacks and existing traffic replay tools. Section 3 describes the controlled testbed and experimental procedure. Section 4 provides a description of RELIVE and its validation by means of different DoS attacks. Section 5 describes the results of our study and an analysis of the key findings. Section 6 discusses the threats to validity and how we mitigated them, while Section 7 concludes the work.

2. Related Work

2.1. Public security datasets and selection criteria

Nowadays, an ever-growing community of researchers and practitioners leverages **public intrusion datasets** to tune and test detection techniques. This choice is justified by the usability of the datasets, which are accessible in different data formats. For example, they might be available in a raw format, such as `pcap` packet data files, more "refined" formats, such as network flows organized in comma-separated values (`csv`) files –specially crafted to apply modern machine learning techniques– or both. Undoubtedly, the availability of such data makes the evaluation of different intrusion detection systems extremely simple.

Many public security datasets have been proposed over the years [6]. Some of them have gained an extraordinary popularity and are still used as a common benchmark, despite the scarce representativeness due to the absence of data attributable to modern attacks. One of these is KDD-CUP'99[1]. The dataset was created in 1999 and used for the Third International Knowledge Discovery and Data Mining Tools Competition. It includes a wide variety of intrusions simulated in a military network environment and has been used in many studies during last 20 years; the reference article for KDD-CUP'99 preparation [14] has been cited around 1200 times according to Google Scholar at April 2021. For years, it has been considered the reference dataset to test most detection algorithms; even recent work, such as [15] and [16], use this dataset for tuning the detection algorithms. After about two decades, the KDD-CUP'99 dataset is hardly ever a perfect representative of present-day networks. These considerations are also valid for the younger NSL-KDD dataset [7], suggested in 2009 to solve some of the inherent problems of KDD-CUP'99, such as biased classification due to over-correlated features.

In recent years, studies that look at security datasets with more critical thinking are spreading. For example, an investigation of the reliability of KDD-CUP'99 is reported in [17]. In particular, the Authors identify some statistical flaws that might introduce bias when training intrusion detection models. A security dataset that faced some criticisms not long after its release is DARPA [18]. It was created, in its different versions (1998-1999), at

[1] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

the MIT Lincoln Lab by collecting five weeks of network traffic in an emulated network environment and by including different types of attacks. In [8] the Authors show that the data provided by DARPA is inappropriate for simulating actual network environments. CICIDS2017 [13] is a security dataset that is gaining a strong popularity. It was released in 2017 and publicly available for researchers; it includes benign traffic and many recent attacks. The extraordinary diffusion of CICIDS2017 is also due to its structure and organization. Its Authors offer a complete suite of tools and resources, ranging from `pcap` to `csv` files that provide ready-to-use labeled flows for those who want to apply machine learning techniques. In addition, Authors provide CICFlowMeter [19], which allows to produce network flows from raw `pcap` files. Other popular datasets that contain non-obsolete attacks are UNSW-NB15 [20], ISCXIDS2012 [21], UGR'16 [22]. The interested reader is referred to [6] for a survey of literature on intrusion detection datasets.

Datasets selection. In the context of this paper we assess five datasets: CICIDS2017, ISCXIDS2012, NDSec-1 2016, MILCOM 2016 and SUEE 2017. All these datasets will be presented in Section 5.1 along with related DoS attacks. The datasets were selected because –differently from others– they provide network traffic data collected under attacks in the form of `pcap` files, which we need to "relive" the attacks by means of our replay tool.

2.2. Denial of Services: background and literature

Denial of Service (DoS) attacks pose a significant threat to the availability of network services [9]. In the broad sense of the term, a DoS attack aims to cause the unavailability of the victim system to legitimate users [23].

Starting from this common purpose, there are many DoS variants that give the attack different shapes and features. Over the years, DoS surveys and taxonomies, such as [24], have been proposed. A first conventional and coarse-grained classification of DoS attacks involves the identification of two categories: **bandwidth depletion attacks** and **resource depletion attacks** [25], [24]. The former aims to consume the entire *bandwidth* of the victim system with unwanted traffic, while the latter aims to consume all the *resources* of the victim, such as memory, sockets or CPU time. It should be noted that while in *bandwidth depletion attacks* any additional malicious message on the network contributes to the effectiveness of the attack (consuming bandwidth), the same does not happen for *resource depletion attacks*. In fact, a message may not be able to consume additional resources on the target server (note that many defense mechanisms work exactly this way, preventing useless consumption of resources). In order to accomplish *resource depletion attacks*, the attackers can follow two different approaches. In particular, they might exploit network, transport and application layer protocol vulnerabilities to hit the victim *(protocol exploit attack)* or send malformed packets with the aim to mislead the victim and crash the system *(malformed packet attack)* [24].

In our paper the focus is on DoS **protocol exploit attacks**, which exploit the weaknesses of network layer protocols as Transmission Control Protocol (TCP) or some application layer protocols as Hypertext Transfer Protocol (HTTP). Typical attacks belonging to this category are: *TCP SYN*, *HTTP Flood* and *Slow HTTP* attacks [24]. *TCP SYN* (also known as SYN flood) exploits part of the normal TCP three-way handshake to consume resources on the targeted server by making it unresponsive. An *HTTP flood* attack, instead, is designed to overwhelm a targeted server with many HTTP requests. Finally, *Slow HTTP* attacks slowly consume all of the resources of the victim. *TCP SYN*, *HTTP Flood* and *Slow HTTP* are the attacks contained in most of the public datasets that we selected for our experiments.

Modern machine learning techniques have been shown effective in intrusion detection; as such, a wide literature on the detection of DoS attacks has been recently produced [26]. For example, a machine-learning-based DoS detection system is presented in [27]. The Authors use an inference-based approach and the detection rate achieved is 96%. Qu *et al.* [28] propose the statistic-enhanced directed batch growth self-organizing mapping (SE-DBGSOM), a recent model based on self-organizing maps (SOM), for DoS attack detection. The proposal is evaluated on the CICIDS2017 dataset. In order to solve the challenges in DoS detection, Nguyen *et al.* [29] propose an intrusion detection system that exploits a convolutional neural network model. The Authors evaluate the performance of the proposed method using the datasets UNSW-NB15 and NSL-KDD. The results are valuable as compared to the state-of-the-art DoS detection methods. Sacramento *et al.* [30] propose *FlowHacker*, which aims to detect malicious traffic on the top of network flows by capitalizing on unsupervised machine learning and threat intelligence: the approach is validated with both the ISCX public dataset and real data by an Internet service provider. Finally, in [31] the Authors propose a DoS anomaly detector that uses a deep autoencoder as a core component. The problem is treated as a semi-supervised task, and the reference dataset is again CICIDS2017. The concept of *adversarial risk* is spreading widely in computer and network security. In recent years, many solutions that exploit adversarial machine learning techniques have been documented in the literature. In [32], for example, different DoS adversarial attacks are studied in order to bypass two trained ANN classifiers. Adversarial DoS samples are effective and the attack is successful with a few queries.

2.3. Network traffic replay tools

Several network traffic replay tools have been produced for analysis purposes so far. Replay tools can be either *stateful* or *stateless*. A **stateful** tool –as opposite to **stateless**– manages the state of the connections during replay; moreover, the content of replayed packets is adapted to fit the specific network configuration of the system under test. Other tools provide for **payload** "re-generation", while others do not alter the *payload* of the original packets. An additional classification encompasses *trace-based* replay and *statistical-based* replay. **Trace-based** replay replicates the content and timing of previously-collected traffic traces. **Statistical-based** replay, instead, adopts a packet generation processes based on statistical models. The statistical information, such as overall packet frequency and timing between packets, is obtained from the original capture and re-generated traces are similar to the original ones. It is worth noting that *traffic generator* tools as Trex[2] or D-ITG [33] are not intended for replay. These tools aim to generate realistic traffic by replicating a previous capture, but do not establish actual connections with the target servers.

Among existing tools for traffic replay, it is worth to mention `TCPOpera` [34], which implements a *statistical-based* replay approach, conceived for a *stateful* emulation of TCP connections. It analyzes a network trace in order to collect information beforehand; then it creates a statistical model of the identified events and generates synthetic traffic from the model. Another replay tool is `TCPivo` [35]. It is a *stateless* and *trace-based* replay engine designed for high-performance packet replay. `TCPivo` leverages pre-fetching techniques to maintain timing accuracy for high speed traces. It also provides an option to replace the packet payload with *null* padding, with the aim to increase the speed at which the packets can be replayed. `DETER` [36] is a *stateful* replay tool. It is essentially used to make diagnoses, as it replays selected packets in order to reproduce performance issues with low overhead.

[2] `https://trex-tgn.cisco.com`

Tool	Stateful	Payload	Approach
TCPOpera	Yes	original payload	statistical-based
TCPivo	No	re-generated	trace-based
DETER	Yes	re-generated	trace-based
tcpreplay	No	original payload	trace-based
tcpliveplay	Yes	original payload	trace-based
RELIVE	Yes	original payload	trace-based

Table 1: Comparison of the replay tools.

A tool that focuses on web server testing is `Monkey`, which replays web application traffic by emulating the TCP stack. It aims to infer delays caused by the client, the applications, the server, and the network.

Close to RELIVE –the tool proposed in this paper– there are `tcpreplay`[3] and `tcpliveplay`[4], which follow a *trace-based* approach. The former is a command-line tool that uses previously captured traces in `pcap` format. In particular, it replays traffic traces at a desired rate without modifying the transport layer header and the payload of a packet. As a result, `tcpreplay` is stateless and does not support synchronizing TCP sequence numbers and acknowledgements. Although `tcpreplay` replays the traffic towards a server, it *does not* really communicate with the server. The lack of communication with the server is well known by the community[5]. For these reasons, `tcpreplay` proved to be ineffective in our testbed, since we had to deal with bidirectional TCP streams that require synchronization of sequence numbers and acknowledgements. On the other hand, `tcpliveplay` –included in the `tcpreplay` suite– was designed to overcome this issue. It statefully replays packet captures by generating updated TCP sequence numbers and acknowledgments. The use of `tcpliveplay` in our testbed has not produced the expected outcome. The existence of issues precluding the proper functioning of the tool is also confirmed and highlighted by ongoing bug fixing activity moved by the community[6].

Table 1 summarizes the main features of RELIVE with respect to existing

[3] https://tcpreplay.appneta.com
[4] https://tcpreplay.appneta.com/wiki/tcpliveplay-man.html
[5] https://stackoverflow.com/questions/37648135
[6] https://github.com/appneta/tcpreplay/issues/540

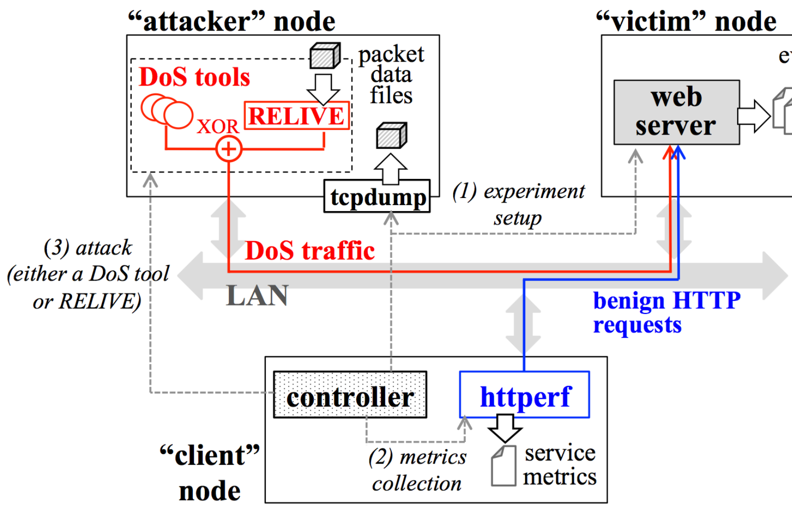


Figure 1: Experimental testbed.

tools. We follow a *stateful* replay approach starting from traffic data available in `pcap` files. At the time being, it is conceived as a *lightweight* and *ready-to-use* solution for replaying and assessing DoS attack traces contained in public intrusion detection datasets.

3. Testbed and Analysis Method

Our study is based on direct measurements with a victim web server during DoS attacks. The victim is monitored during the progression of the attacks in order to collect a variety of service metrics. In the following we present the experimental testbed, the service metrics and a capacity analysis aiming to properly tune the experiments. The testbed hinges on RELIVE, which will be thoroughly described and validated in Section 4.

3.1. Experimental Testbed

Experiments are conducted in a private network infrastructure at the University of Sannio. The experimental testbed consists of three Ubuntu 18.04 LTS nodes, equipped with Intel Xeon E5-2650V2 8 cores (with multithreading) 2.60 GHz CPU and 64 GB RAM within a local area network (LAN) over 56Gb/s Infiniband. Nodes and experimental procedure are described according to Figure 1.

The **“victim” node** hosts an installation of the Apache web server 2.4.29, which is a significant case study, given its widespread use. It is currently adopted from personal blogs to websites serving a large base of users; moreover, it is a typical attack target in many public intrusion datasets. This web server supports a variety of modules –including security-related capabilities– that can be enabled/disabled by suitable configuration of the *baseline* server installation. Among the variety of modules, ***reqtimeout***[7] is strongly pertinent to the context of this study. According to the documentation, the module is available since Apache HTTPD 2.2.15 –thus April, 2010[8]– and allows to set timeouts and minimum data rates for receiving requests. The module can mitigate some DoS attacks and is typically enabled *by default* in the baseline server after installation from the standard Ubuntu repository, which means that its disablement requires explicit changes of the configuration by the user. As shown later on in this paper, we conduct the experiments both in case of *no* and *with reqtimeout* enabled for the sake of comprehensive evaluations. In this respect, it is worth noting that the authors of *none* of the datasets assessed in this study make it clear whether *reqtimeout* (or any other defense mechanism) was enabled at the time the attacks were conducted.

The **“attacker” node** is intended to generate DoS traffic data aiming to disrupt server operations. The node underlies two **usage modes**:

- *attack emulation*: direct emulation of DoS attacks by means of state-of-the-art tools;
- *attack replay*: replay of prerecorded attack traffic from a previous capture available in a *packet data file* by means of RELIVE.

Usage modes follow an *exclusive OR* (i.e., XOR in Figure 1) policy, which means that –at a given time– either there is no DoS traffic at all through the LAN or, if any, it is generated exclusively in one of the modes. The attacker node features also an instance of `tcpdump`, which is used to capture the traffic between the attacker and the victim in a packet data file. We rely on a mixture of *attack emulation* and *attack replay* to demonstrate the effectiveness of RELIVE in Section 4, while existing datasets are assessed through *attack replay* in Section 5.

[7] https://httpd.apache.org/docs/2.4/mod/mod_reqtimeout.html
[8] https://archive.apache.org/dist/httpd/

The **"client" node** hosts `httperf`[9] and the *controller*. The former is a well-known load generator. It is used here to probe the web server by gathering convenient service metrics that summarize its operational status. The latter, i.e., the **controller**, automates and orchestrates the execution of the experiments, which consist of the following steps, shown in Figure 1:

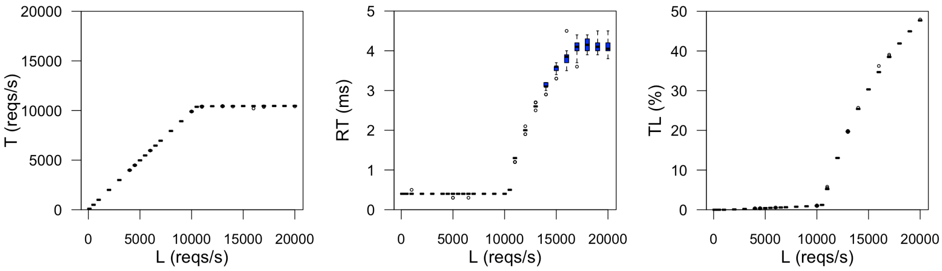
1. *experiment setup*: cleaning up the logs of the web server (i.e., *access* and *error log* of the server), boot of the web server and `tcpdump`;
2. *metrics collection*: generation and storage of the service metrics by means of `httperf` during the progression of the attack;
3. *attack*: execution of a DoS attack by either a dedicated tool or reliving a previous capture; the web server is now exercised with *benign* HTTP requests from `httperf` –referred to as the *load* in the following– and DoS traffic;
4. *experiment completion* (not represented in Figure 1 for better readability): shutdown of either the attack tool or RELIVE, `httperf`, `tcpdump` and web server, storage of the packet data file, service metrics and event logs for subsequent analysis, reboot of the nodes to ensure independent experimental conditions prior the next run.

It is worth pointing out that the hardware and software of our testbed are more recent and possibly powerful than the ones used for dataset collection a few years ago. Depending on the type of attack performed, this may contribute to higher server availability and lower response times than in the "original" environment.

3.2. Service Metrics and Capacity Analysis

The controller continuously runs `httperf` to probe the web server under attack and to collect service metrics at regular intervals of time. Noteworthy, `httperf` makes it possible to set a desired level of *load* consisting of HTTP requests in order to exercise the target server. The **load (L)** submitted to the server is measured here in HTTP requests per second (*reqs/s* in the following). In response to the load, `httperf` generates several convenient metrics. In this study we focus on (i) **reply rate** or **throughput (T)**, i.e., HTTP requests accomplished by the web server within the time unit (measured here again in *reqs/s*) and (ii) **response time (RT)**, i.e, mean

[9] `https://github.com/httperf/httperf`



(a) throughput (T) (b) response time (RT) (c) throughput loss (TL)

Figure 2: Capacity analysis of the web server.

time taken to serve a request measured in milliseconds (*ms*). We complement the analysis by computing a derived metric, i.e., **throughput loss (TL)**, as follows:

$$TL = \frac{L - T}{L} \cdot 100 \qquad with\ L \geq T\ and\ L > 0 \tag{1}$$

TL is the percentage of HTTP requests out of the incoming load L that are not accomplished by the web server within the time unit. Differently from T, TL varies within [0,100]%.

We conduct a **capacity analysis** of the web server in order to determine the *maximum load* that can be handled by the server. Later on in the experimental section of the paper, this is useful to assure that a potential performance loss of the server is caused by a DoS attack rather than an accidental circumstance caused by badly tuned load. During the capacity analysis the web server is exercised solely by means of `httperf`, which means there is no attack activity. We measure T, RT and TL obtained in response to increasing values of L: for each level of L we execute 30 repeated runs of `httperf` to collect a statistically significant sample of the metrics.

The *knee capacity* [37] of the server in our testbed is reached around L=10,000 *reqs/s*, as shown in Figure 2a, where the throughput stops growing linearly as a function of L. After the knee, the throughput saturates and T<L: accordingly, both RT and TL increase rapidly as a function of the load, as shown in Figure 2b and 2c, respectively: for example, when L=20,000 *reqs/s* we obtain RT=4 *ms* and TL close to 50%. When the web server is free from a concurrent attack and it is operated *below* the knee capacity we expect

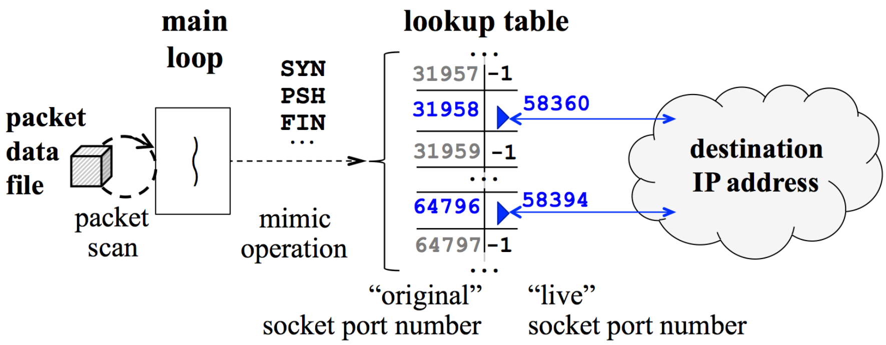


Figure 3: RELIVE: conceptual overview.

T≈L and in turn TL≈0%, which means *no loss* of HTTP requests; on the other hand, a value TL>0% could point out the presence of a DoS attack, because in our controlled testbed the only source of legitimate activity is the "client" node.

4. RELIVE: Implementation and Assessment

Our support tool allows to **relive** –hence its name– previously-captured traffic made available in the form of packets in a file. We use a *stateful* approach by instantiating actual connections towards a desired destination Internet Protocol (IP) address. In the context of this study RELIVE is leveraged to replay DoS attack traffic against the victim web server in the controlled testing environment presented above, making it possible to reproduce and measure its effects at application-level.

4.1. Replay Approach

Figure 3 shows a conceptual overview of RELIVE. TCP packets, which were originally sent to a given destination, are fed to the tool and "relived" towards a user-supplied destination IP address *over brand new sockets and connections*. Runtime information on *live* sockets and connections is maintained through a **key-value** *lookup table* (shown in Figure 3), which stores the mapping between "original" socket port numbers (i.e., the source port of the packets in the file) and "live" port numbers (i.e., the ports of the sockets used to actually relive the traffic). The tool is implemented in `python`, lever-

```
—– start connection packet (SYN) —–
09:07:05.766440  IP 192.168.56.102.39842 > 192.168.56.101.http: Flags [S],
seq 2633020550, win 64240, options [...], length 0
—– push data packet (PSH) —–
09:07:05.766728  IP 192.168.56.102.39842 > 192.168.56.101.http: Flags [P.],
seq 0:19, ack 1, win 502, options [...], length 19: HTTP: GET /?12 HTTP/1.1
```

Figure 4: Example SYN and PSH packets obtained with `tcpdump -r`.

aging `dpkt`[10], i.e., a module for packet creation and parsing with definitions for the basic TCP/IP protocols. The packet format expected by RELIVE is `pcap`, `LINKTYPE_ETHERNET` *link-layer header type*, as the one typically generated by `tcpdump`[11]. It is worth noting that other packet formats, such as `pcap-ng` and `LINKTYPE_LINUX_SLL` *link-layer header type*, can be adapted to RELIVE with minor effort by means of the widely-used utility programs `editcap` and `tcprewrite`, as we did in our experiments.

RELIVE scans the input data file (*main loop*, leftmost part of Figure 3) in order to extract some key fields of the packets, i.e., *timestamp*, *source port*, *flag*, and *data*. Figure 4 shows two example packets obtained by means of `tcpdump`, i.e., start connection (SYN) and push data (PSH); the fields of interest are enclosed in a box for better visualization. At a given time, traffic replay proceeds according to the fields of the packets and the status of the *lookup table*, named `liveSockets` in the following, whose *keys* are positive integers and *values* are all set to `-1` before the beginning of the scan. Given a packet, its **source port**, `srcport`, is used as the key to access `liveSockets`:

- if the value corresponding to the key is `-1` then a new socket is instantiated *on-the-fly* in the case of a `SYN` TCP packet. The socket is connected to the target destination and stored in the *lookup table*, which means a mapping between the "original" and "live" port number has been established, as depicted in Figure 3 (▷ symbol);
- otherwise, i.e., if exists a mapping, RELIVE will mimic the socket operation intended by the TCP packet by means of a suitable `python` method (e.g., `send`, `shutdown`, `close`), such as *sending* data for a `PUSH`

[10] https://dpkt.readthedocs.io/en/latest/
[11] http://www.tcpdump.org/linktypes.html

packet (the data to be sent are extracted beforehand from the packet itself), *finishing* the connection for a `FIN` packet, and so forth.

As for the timestamps that accompany the packets, shown in Figure 4, they are used by RELIVE to regulate the *speed* of the scan. In fact, operations described above *do not* progress at the maximum speed with no awareness of time; rather, they are deferred based on the difference between the timestamp of a given packet and its predecessor in the file. This approach allows RELIVE to emit the packets with the same timing of the original sequence, as recorded in the input data file.

Overall, the stateful replay approach described above capitalizes on the *State behavioral design pattern* [38], where the outcome of the set of operations –represented by the possible values of the TCP *flag* field in our context– depends on the *state* of the mapping, i.e., whether a given value in the table equals `-1` or not, and the *state* varies during the execution based on the occurrence of specific operations. Noteworthy, the *State* design pattern is widely used to specify and program network protocols.

4.2. Empirical Assessment

The functioning of RELIVE is assessed by direct emulation and replay of various DoS attacks. To this aim, we use publicly-available scripts and a command line utility program, which are widely-used by the security community, as the research groups that published the datasets addressed by our paper. Attacks and pertinent information on the corresponding tools are listed below:

- `hulk`: it is conceived as an HTTP flood aiming to overwhelm a given web server by continuously requesting URLs. The strength of Hulk is the ability to produce patterns that cannot be easily detected. The core idea is to generate a unique pattern at *each* and *every* request by increasing the load on the web server. The attack leverages different strategies. One of these is the *obfuscation of source client*. This is accomplished by using a list of known *user agents* and, for every suitably crafted request, the *user agents* is a random value out of the known list. One of the most popular implementations is *grafov* Hulk[12], used for our tests, which is a `python` script also available in `Go` language.

[12] `https://github.com/grafov/hulk`

- `TCP flood`: it is another popular DoS attack and well-known to the community. The attacker sends TCP connection requests locking the available ports on the server and causing incapability to accept legitimate TCP connection requests from other hosts; therefore, it can be considered as flooding attack. For our experiments we used the well-know *Leeon123* TCP flood script[13]. It is a Python script that allows to launch a TCP flood attack against the victim machine in seconds.

- `slowloris`: it allows to launch a *slow DoS* attack against a target server. This class of attacks uses *low-bandwidth* approaches, which exploit a weakness in the management of TCP fragmentation of the HTTP protocol: it requires HTTP messages to be completely received before they are processed. In order to accomplish this attack, we used the *gkbrk* slowloris script[14]. After having established a number of connections to the target server, the script keeps them alive as long as possible. This task is accomplished by sending keep-alive headers on all connections at 15 s intervals. If the server closes a connection, this is then restored by keeping constant the total number of connections.

- `slowhttptest`: it is a highly configurable tool that can be used to generate *slow DoS* application-layer attacks[15]. We use `slowhttptest` in the "slowloris" mode, which allows to send incomplete HTTP requests to the target server. Both `slowloris` and `slowhttptest` implement a slow attack. We use both, in order to validate RELIVE with respect to different implementations of the same base behavior. It is worth noting that the results produced by the two tools –shown in the following– are different and that RELIVE correctly reproduces both of them.

- `SYN flood`: it is one of the most known attacks for TCP stacks, which capitalizes on a weakness of the TCP handshake. The weakness is due to the fact that the server allocates resources *before* the client: in consequence, a client may forge multiple malicious packets (potentially spoofing IP addresses) in order to cause the victim server to allocate a large amount of resources (as socket ports and memory). At the state-of-the-art, many operating systems, in particular Linux, use the

[13] `https://github.com/Leeon123/TCP-UDP-Flood`
[14] `https://github.com/gkbrk/slowloris`
[15] `https://tools.kali.org/stress-testing/slowhttptest`

syn-cookie technique [39] as a countermeasure; this is typically applied as a default by the kernel, making the most part of real-world servers protected by the attack. The attack is emulated with a public `python` script[16]; moreover, we had to purposely disable the *syn-cookie* capability[17] in order to make the attack effective against the server.

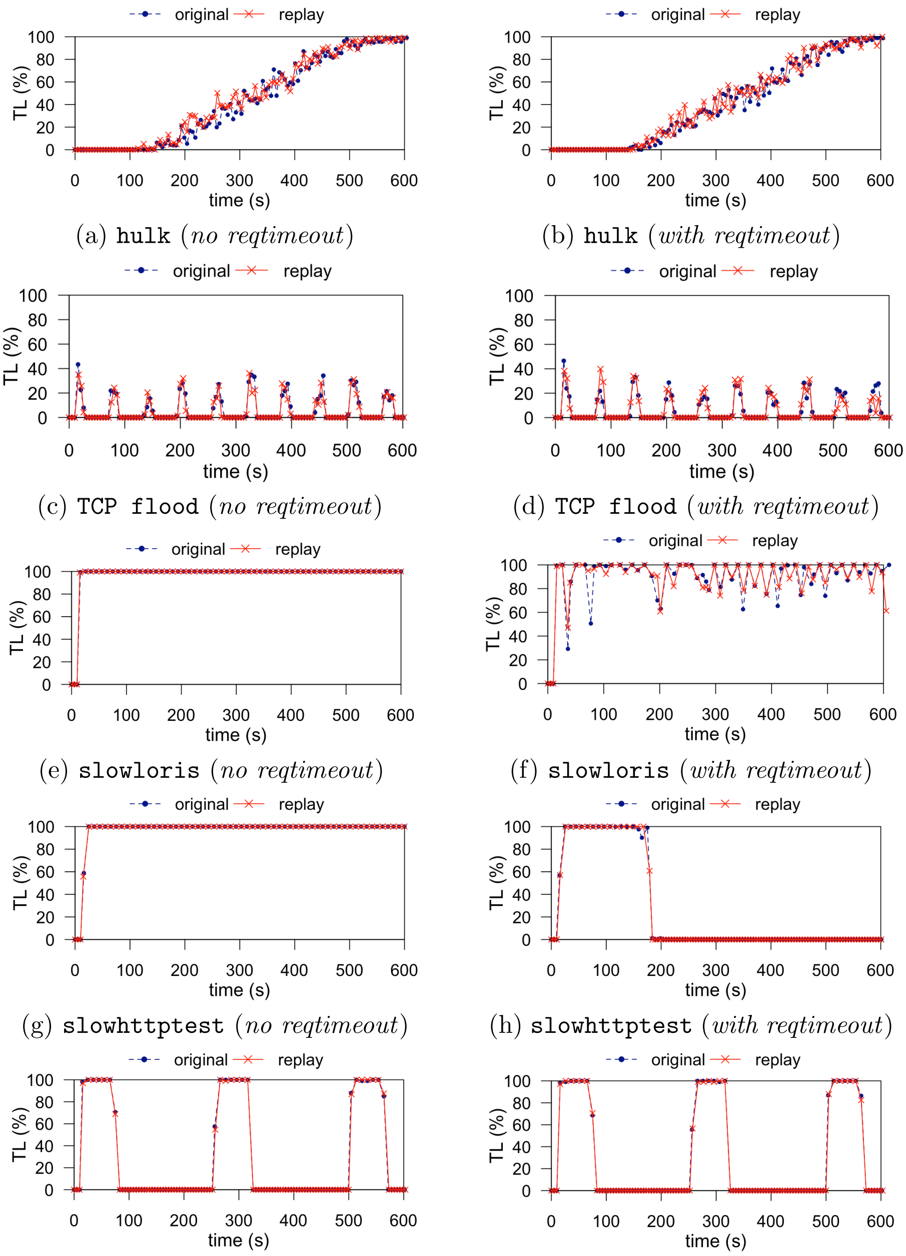
Overall, the attacks underlie a mixture of different DoS protocol exploit attacks; more important, they elicit quite different outcomes by the victim web server, as shown in Section 4.2.1. For each attack listed above, we conduct two independent experiments. The former (denoted by "**original**" in the following) consists in performing the attack against the web server while capturing the network packets in a `pcap` data file (*attack emulation* mode in Section 3.1); the latter (denoted by "**replay**" in the following) is done by replaying the `pcap` file –thus the by-product of the former experiment– by means of RELIVE (*attack replay* mode in Section 3.1). Moreover, each attack and its corresponding replay are run both in case of *no* and *with* the *reqtimeout* module enabled, which sum up to total 10 original-replay *paired* experiments. The duration of each experiment is set to 600 *s*, time interval which is long enough to collect a large sample of service metrics generated by `httperf`, as described in Section 3.1. Moreover, in all cases the attack starts at t=15 *s* since the beginning of the experiment and the web server is exercised with a load of L=1,000 *req/s*, rate that can be safely handled at no TL according to the capacity analysis presented in Section 3.2.

4.2.1. Analysis of the Throughput Loss (TL)

Figure 5 shows how TL varies during the progression of the attacks in case of *no* (Figure 5a, 5c, 5e, 5g, 5i) and *with* (Figure 5b, 5d, 5f, 5h, 5j) *reqtimeout* enabled; the x-axis represents the time since the beginning of the experiment. Each plot provides TL for the *original* attack (•-*marked* series); moreover, we superimpose TL obtained with the corresponding *replay* of the attack (×-*marked* series) for better visualization purposes.

It can be noted that the attacks cause a variety of responses by the victim web server, which allow to gain insight into the effectiveness of RELIVE in face of different operating conditions. For example, `hulk` (Figure 5a and 5b) appears to be silent up to 150 *s* until it saturates the server by the end of the

[16] `https://github.com/Leeon123/Simple-SYN-Flood`
[17] `https://nixcp.com/enable-tcp-syn-cookie-protection-linux/`



(a) `hulk` (*no reqtimeout*) (b) `hulk` (*with reqtimeout*)

(c) `TCP flood` (*no reqtimeout*) (d) `TCP flood` (*with reqtimeout*)

(e) `slowloris` (*no reqtimeout*) (f) `slowloris` (*with reqtimeout*)

(g) `slowhttptest` (*no reqtimeout*) (h) `slowhttptest` (*with reqtimeout*)

(i) `SYN flood` (*no reqtimeout*) (j) `SYN flood` (*with reqtimeout*)

Figure 5: Throughput Loss (TL) for each attack and its replay.

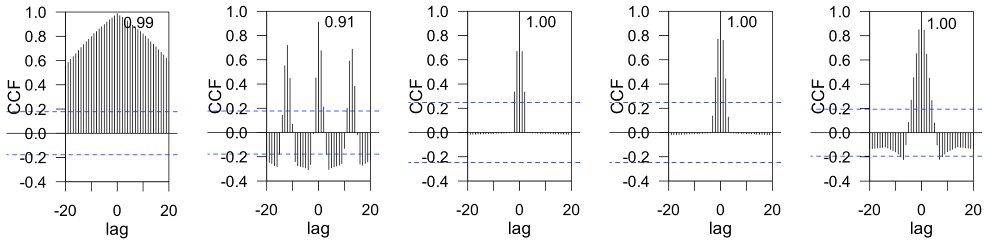
experiment, where TL approaches 100%. `TCP flood` (Figure 5c and 5d) is characterized by periodic spikes with a maximum TL=43.4%. At the other end of the spectrum, both `slowloris` and `slowhttptest` show the typical "on-off" behavior of low-bandwidth attacks in lack of proper defense (*no reqtimeout* in Figure 5e and 5g, respectively): in fact, TL abruptly reaches 100% as soon as the server is saturated by the attack; similar considerations hold for the `SYN flood` attack, which we launched for three times during the progression of the experiment at t = 15 *s*, 256 *s* and 504 *s*. As for the impact of the *reqtimeout* module, we observe that it is ineffective for `hulk`, `TCP flood` and `SYN flood`, while it mitigates slow attacks at some extent, as can be noted in Figure 5f and 5h. The most interesting outcome is that the TL measured during the progression of a replay experiment is close to the corresponding original attack both for *no* and *with reqtimeout* runs.

The visual test, which provides exploratory insights into the capability of RELIVE at correctly "reliving" previously-captured attacks, is supplemented here by more robust statistical analysis. TL values measured for each original attack and its *paired* replay experiment are seen as two time series consisting of n observations, namely o_t and r_t, respectively. In order to model the relationship between o_t and r_t, we compute the **cross correlation function (CCF)**, i.e., the set of correlations between o_t and $r_{(t+h)}$ that are obtained by varying the integer h, i.e., the *lag* between the series, in $-n<h<+n$. Eq. 2 shows how the CCF of o_t and r_t is computed for a given lag h. The numerator, i.e., $c_{o,r}(h)$, is the *cross variance* of o_t and r_t for the lag h; it is divided by the product of the standard deviations of o_t and r_t, i.e., σ_o and σ_r, which are the square roots of their respective variances $c_{o,o}(0)$ and $c_{r,r}(0)$ in Eq. 2.

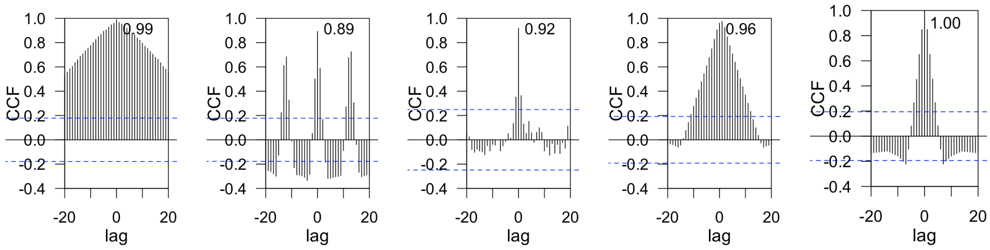
$$CCF_{o,r}(h) = \frac{c_{o,r}(h)}{\sqrt{c_{o,o}(0)c_{r,r}(0)}} = \frac{c_{o,r}(h)}{\sigma_o \sigma_r} \tag{2}$$

We obtain $-1 \leq$CCF$\leq +1$. Intuitively, the relationship between the time series is deemed *strong* if $|$CCF$| \geq 0.80$, *moderate* if $0.50 < |$CCF$| < 0.80$ and *weak* if $|$CCF$| \leq 0.50$ [40].

Figure 6 shows the CCFs for each pair of series in Figure 5 with $-20 \leq h \leq +20$ by step 1; horizontal dashed lines denote the values beyond which the correlations are statistically different from zero. For example, Figure 6a shows the CCF corresponding to the series in Figure 5a; the same correspondence applies for all the pairs of plots with the same caption. It should be



(a) `hulk` (*no reqtimeout*) (b) `TCP flood` (*no reqtimeout*) (c) `slowloris` (*no reqtimeout*) (d) `slowhttptest` (*no reqtimeout*) (e) `SYN flood` (*no reqtimeout*)



(f) `hulk` (*with reqtimeout*) (g) `TCP flood` (*with reqtimeout*) (h) `slowloris` (*with reqtimeout*) (i) `slowhttptest` (*with reqtimeout*) (j) `SYN flood` (*with reqtimeout*)

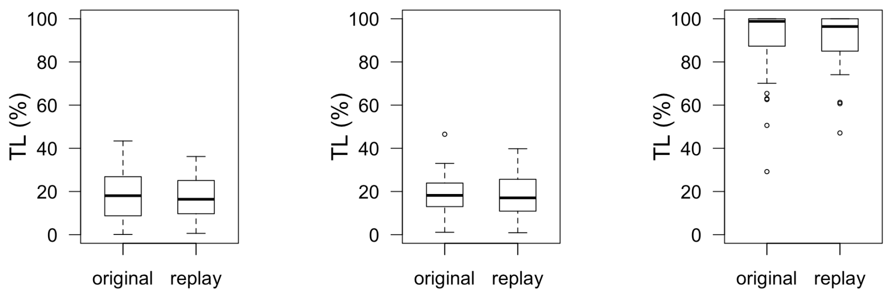
Figure 6: Cross correlation function (CCF) of each attack and its paired replay experiment.

noted that we aim to demonstrate that original and replay TL series happen *with the same timing* since the beginning of the experiment: this information is summarized by the CCF value computed at **lag=0**, which correlates o_t and r_t with no lag (i.e., h=0), and is annotated in the top-right corner of each plot in Figure 6. In all cases the CCF is significantly higher than 0.80: as such, correlation between the time series is *strong*. As for the remaining lags in Figure 6, they are intended to show that the maximum CCF occurs *exactly* at lag=0, which means the replay is not behind/ahead the original attack.

Other typical **statistical characterizations** encompass *sample mean*, *standard deviation* and *95% confidence interval* (CI) of the TL observations, which are shown in Table 2 for a subset of the attacks where this type of characterization is more suited (e.g., no "on-off" behavior). In all cases, it can be noted that the sample mean of TL during the original attack is *within*

Table 2: Summary of TL statistics for a subset of the attacks and their replay.

		mean	**standard deviation**	**95% confidence interval**
`TCP flood`	original	18.0	10.7	(14.7, 21.4)
(no reqtimeout)	replay	16.9	10.2	(13.7, 20.1)
`TCP flood`	original	17.9	9.6	(15.0, 20.9)
(with reqtimeout)	replay	18.4	10.5	(15.1, 21.6)
`slowloris`	original	90.9	14.3	(87.3, 94.6)
(with reqtimeout)	replay	91.4	11.6	(88.4, 94.4)



(a) `TCP flood` (*no reqtimeout*) (b) `TCP flood` (*with reqtimeout*) (c) `slowloris` (*with reqtimeout*)

Figure 7: Paired TL boxplots for a subset of the attacks and their replay.

the CI of the replay and viceversa, which means that the samples are not statistically different. This finding is further confirmed in Figure 7, where it can be noted that the boxplots of the TL observations strongly overlap (○-marked points denote measurement outliers). Overall, given the CCF and the statistical characterization above, it can be reasonably claimed that the impact of the original attack and its replay at the application-level *are not* statistically different with respect to the set of attacks in hand.

4.3. Non-Goals of the Current Implementation

At the current stage of development, RELIVE is strongly driven by the objectives of the paper and our current research. Here the focus is on a *core set* of capabilities that allow to reproduce the significant class of DoS protocol exploit attacks. Accordingly, **non-goals** of our current implementation include UDP-based attacks, further protocols beside HTTP or replication of documented vulnerabilities, just to mention a few. While RELIVE is indeed capable to replay TCP traffic, application-level "effects" of the replay ultimately depend on the protocols in hand, which require dedicated design. For example, an attempt to replay SSH traffic will likely lead to authentication issues. Regarding vulnerabilities –also related to DoS– there exist specialized databases and tools, such as IXIA Perfect Storm[18]. Given the ever-increasing diversity of attacks, victim applications and operating conditions, RELIVE will serve as a long-term toolset with the potential for further developments and usage modes that go beyond the scope of the current paper.

5. Experimental Results

We present the analysis of DoS traffic from public datasets. Traffic is replayed under different combinations of key factors in our testbed according to the *attack replay* mode presented in Section 3.1. Assessed datasets, results and practical implications of the findings are presented in the following.

5.1. Datasets description

The main features of the datasets are presented below. The interested reader is referred to the link and reference paper reported for each dataset.

CICIDS2017[19] is a public dataset proposed by the research team of the Canadian Institute for Cybersecurity, which includes both normal traffic and attacks at the state-of-the-art when data were collected [13]. The dataset is available both in packet format (`pcap`) and bidirectional flow labeled format (`csv`). The data capture period started at 9 a.m., Monday, July 3, 2017 and ended at 5 p.m., Friday, July 7, 2017, for a total of 5 days. *Monday* is the "normal day" and contains only benign traffic; DoS attacks addressed by our paper, such as `hulk`, `slowloris` and `slowhttptest` belong to the capture of *Wednesday*, i.e., the "DoS day". The attacker was a Kali Linux node and the victim an Ubuntu 16.04 system with an Apache web server.

ISCXIDS2012[20] is a public dataset for intrusion detection purposes, providing normal and malicious network traffic [21]. It is available in packet-based (`pcap`) and bidirectional flow-based (`xml`) formats. ISCXIDS2012 was created by capturing traffic in an emulated network environment over one week. In particular, the capture period runs from 11 June 2010 to 17 June

[18] `https://www.ixiacom.com/products/perfectstorm`
[19] `https://www.unb.ca/cic/datasets/ids-2017.html`
[20] `https://www.unb.ca/cic/datasets/ids.html`

2010. The normal activity was captured on 11, 12 and 16 June; on the other days –in addition to the normal traffic– the dataset contains different attacks. Since our focus is on DoS, we use the data collected on June 14, 2010, referred to in the dataset as *HTTP DoS*. The victim server was an Ubuntu 10.04 system with an Apache web server (version 2.2.9).

NDSec-1 2016[21] is a public-domain dataset, designed in 2016 as an attack composition for network security and proposed by the Network and Data Security Group (NDSec) of the Fulda University of Applied Sciences [41]. It provides packet traces (`pcap`) as well as log files documenting the ground truth and relies on bidirectional flows captured using the flow exporter YAF. For our tests we selected *HTTP flood* and *SYN flood* attacks; HTTP floods were carried out using the `Apache HTTP server`[22] benchmarking tool.

MILCOM 2016[23] is a public-domain dataset, generated for the Applied Communication Sciences' MILCOM 2016 paper [42]. The dataset was created to support reproducible research experiments that address security challenges. The dataset, available in `pcap` format, is arranged in 5 subcategories: A, B, C, D, E. The malicious activity for datasets A through D consists of malware implant and the operation of the ACS pseudo botnet. The malicious activity for dataset E comprises DOS attacks, including the *slowloris* and *SYN flood* used for our experiments.

SUEE 2017[24] is a public-domain dataset which contains both benign and malicious traffic relative to the web server of the Student Union for Electrical Engineering at Ulm University [43]. Released in 2018, the SUEE 2017 dataset is distributed in `pcap` format. It is worth pointing out that the dataset is not labeled; however, the attacker IP ranges are clearly disclosed by the proposing authors, which allowed us to identify DoS traffic. The dataset contains *slow attacks*.

5.2. Initial Experiments

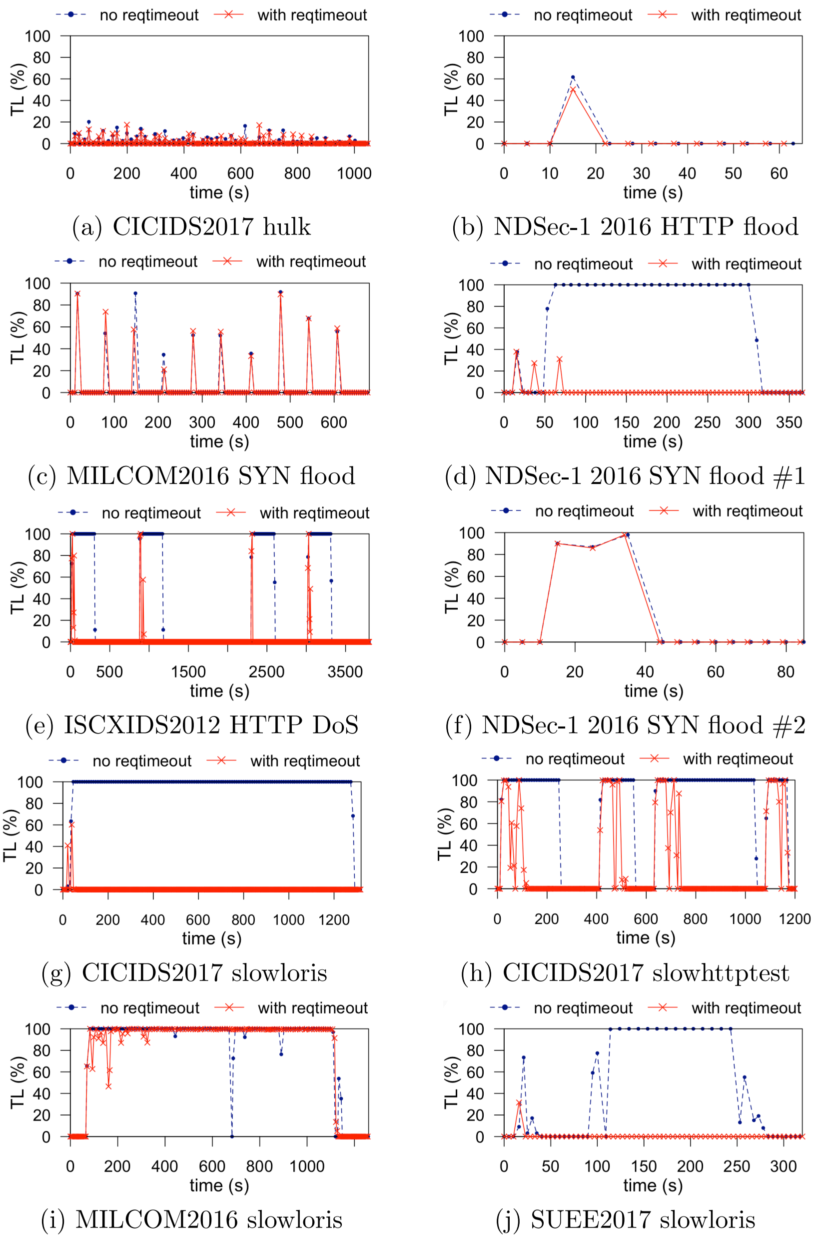
We use RELIVE to replay DoS traffic data from the public datasets listed above against the web server in the controlled testbed in Figure 1 (*attack replay* mode in Section 3.1). Each attack is replayed twice, with two independent experiments: *no* and *with* the *reqtimeout* module enabled at the

[21] https://www2.hs-fulda.de/NDSec/NDSec-1/Files/

[22] http://httpd.apache.org/docs/2.4/programs/ab.html

[23] https://www.netresec.com/?page=ACS_MILCOM_2016

[24] https://github.com/vs-uulm/2017-SUEE-data-set



(a) CICIDS2017 hulk

(b) NDSec-1 2016 HTTP flood

(c) MILCOM2016 SYN flood

(d) NDSec-1 2016 SYN flood #1

(e) ISCXIDS2012 HTTP DoS

(f) NDSec-1 2016 SYN flood #2

(g) CICIDS2017 slowloris

(h) CICIDS2017 slowhttptest

(i) MILCOM2016 slowloris

(j) SUEE2017 slowloris

Figure 8: TL under DoS attacks against the web server.

server-side, respectively. The load generated by the "client" node –aimed to probe the operational status of the server and to infer the service metrics– is set to L=1,000 *req/s*. It is worth noting that this L rate is intentionally much lower than the knee capacity of the server, in order to assess the impact of the attacks. Further considerations on increasing loads are presented in Section 5.4.

Figure 8 shows how TL varies for the attacks. For each plot –corresponding to a given dataset and attack– the TL series obtained during the *no reqtimeout* experiment (•-*marked* series) is superimposed to *with reqtimeout* (×-*marked* series): as such, the reader can compare the impact of the same traffic data in face of different defense scenarios. Attacks start at t=15 *s* since the beginning of the experiment. Overall, each attack has its own peculiarities, such as impact on the throughput, duration and burstiness. For example, CICIDS2017 hulk (shown in Figure 8a) causes *low* TL, i.e., the sample mean is 5.21 within [4.06, 6.36] at 95% confidence for *no reqtimeout*. However, it is not affected by the defense module, where TL has a mean equal to 5.38, within [4.25, 6.51] at 95% confidence, and so it is not statistically different from the former run. On the other hand, NDSec-1 2016 SYN flood #1 (shown in Figure 8d) is strongly effective, i.e., TL=100% through all the progression of the attack, but it suffers from *reqtimeout*, which –if enabled– is capable to almost suppress the attack. As for the slow attacks in Figure 8, they all appear effective at saturating the victim server in case of *no reqtimeout*, i.e., TL=100%, such as Figure 8g and 8h; out of them, MILCOM 2016 slowloris is the only slow attack that remains effective with *reqtimeout* enabled.

Finding: DoS attack traffic provided by public intrusion detection datasets suffers from the presence of *reqtimeout*, which makes the attacks much less effective, if not ineffective at all.

This point is crucial. Again, *reqtimeout* is a *default* module enabled at the installation of the web server and we had to "manually" edit the configuration to exclude the module and to perform the *no reqtimeout* experiments. On the other hand, the research groups that published the datasets make it clear *for none of the attacks* whether they manually checked for the presence of *reqtimeout* –or any other similar defense module– embedded by the installation of the victim servers or the operating system. It is worth noting that, similarly to *reqtimeout*, we had to manually disable the *syn-cookie* capability of the kernel in order make *SYN flood* attacks effective against the server; however, this aspect is not clearly stated in any of the datasets. Inadvertent

Table 3: Configuration parameters of the web server.

parameter	default configuration	adjusted configuration
`StartServers`	2	16
`MinSpareThreads`	25	75
`MaxSpareThreads`	75	150
`ThreadLimit`	64	2,048
`ThreadsPerChild`	25	2,048
`MaxRequestWorkers`	150	4,096

inclusion of defense mechanisms during traffic collection can strongly bias the data released to the research community. In practice, an attempt to learn intrusion detection patterns on top of these data may lead to incorrect outcomes because the behavior of a given attack depends on the specific defenses in hand.

5.3. Impact of the Configuration

According to the initial set of experiments presented above, all the attacks are "apparently" effective in case of *no reqtimeout*. Nevertheless, there is one more key aspect –not purposely touched so far– that is strongly overlooked by the literature on public datasets: the *configuration of the victim server*. Beside regulating the presence or not of supplemental modules for hardening the installation of a given web server, the **configuration** makes it possible to set many other crucial parameters.

For example, in a typical Linux-based installation of the Apache web server the configuration can be accessed in several files in the `/etc/apache2/` directory. The middle column of Table 3 shows the **default values** of key operational parameters that we found after having installed the web server by means of `apt-get install apache2` pointing to the standard Ubuntu repository[25]. It should be noted that the research groups that published the data assessed in our study *do not* concretely touch and disclose configuration-related aspects. Since not stated otherwise, it is reasonable to assume that

[25] `http://it.archive.ubuntu.com/ubuntu` bionic-updates/main amd64 Packages

they conducted the attacks –and collected the related data– with the default configuration of the server. The default configuration might not necessarily reflect "real-life" production servers intended to handle a large base of users; as such, it is worth investigating whether the effectiveness of the attacks depends on the actual configuration.

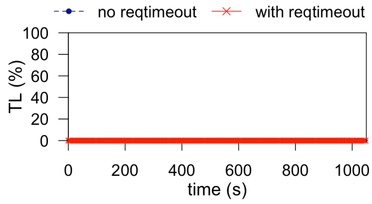
We edited the configuration of the web server in order to significantly boost its capacity and multithreading capability, by raising parameters, such as *start servers*, *thread limit* and *maximum workers*. The result is an **adjusted configuration**, whose settings are shown in the rightmost column of Table 3. Figure 9 shows how TL varies during the progression of the attacks against the web server in case of the adjusted configuration. Analogously to the experiments in Section 5.2 –conducted with the *default configuration*– each attack is replayed twice, i.e., *no* and *with reqtimeout*, and the server is probed with a "client" load L=1,000 *reqs/s* in order to gather the service metrics. Surprisingly, in many cases the same set of DoS attacks does not affect at all the throughput of the victim server. For instance, TL=0% steadily in Figure 9a, 9e or 9j and other similar examples. Only the attacks of MILCOM2016 (Figure 9c and 9i) and one SYN flood from NDSec-1 2016 (Figure 9f) are effective to some extent in case of *no reqtimeout*, with TL close to 100% at several points. After the activation of *reqtimeout*, we found out that only two attacks across all the datasets are still effective (Figure 9c and 9f – *with reqtimeout* series).

Finding: The configuration of the victim server has a major impact on the actual effect of a DoS attack. Public traffic data gathered by executing an attack against the default configuration of the victim server might not be representative to infer general lessons on the resilience of real-life servers operated with optimized configurations.

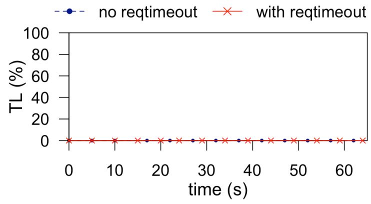
We hypothesize that the research groups that published the datasets observed an actual performance slowdown of the victim servers in response to attacks: as such, they decided to release the data. However, it is reasonable to state that, in many cases, the slowdown was caused by poorly configured servers rather than well-crafted attacks. Because of this limitation, the usefulness of public DoS traffic data to drive application-level *security* claims on servers or intrusion detection systems is questionable.

5.4. Considerations on the Load

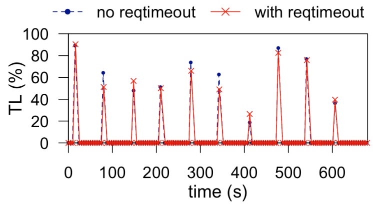
We complement the analysis with an additional set of experiments aiming to account for the **load** generated by the "client" node. In fact, as well



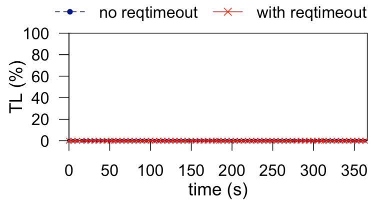
(a) CICIDS2017 hulk



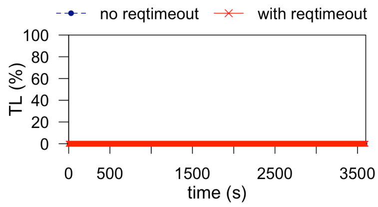
(b) NDSec-1 2016 HTTP flood



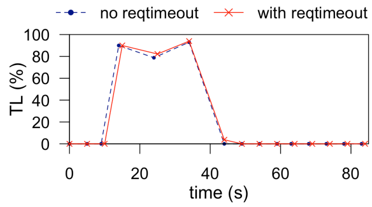
(c) MILCOM2016 SYN flood



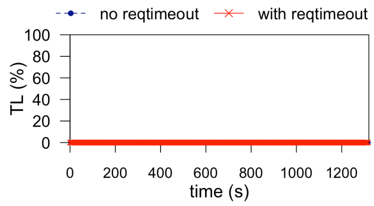
(d) NDSec-1 2016 SYN flood #1



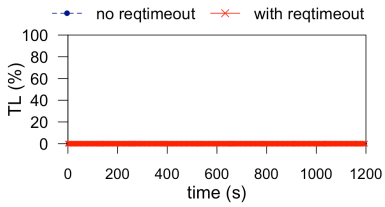
(e) ISCXIDS2012 HTTP DoS



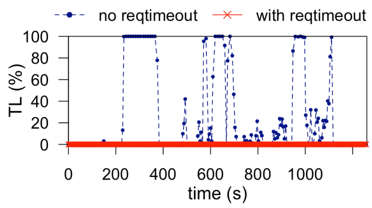
(f) NDSec-1 2016 SYN flood #2



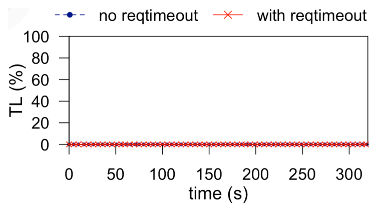
(g) CICIDS2017 slowloris



(h) CICIDS2017 slowhttptest

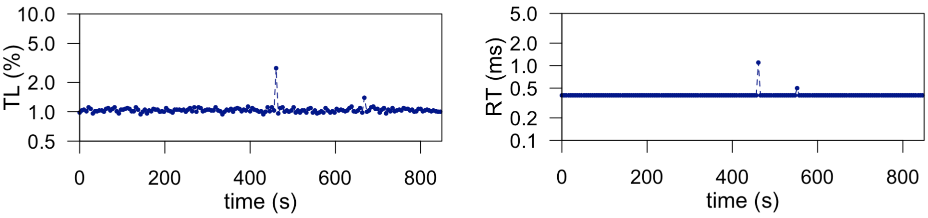


(i) MILCOM2016 slowloris



(j) SUEE2017 slowloris

Figure 9: TL under DoS attacks against the web server (adjusted configuration).



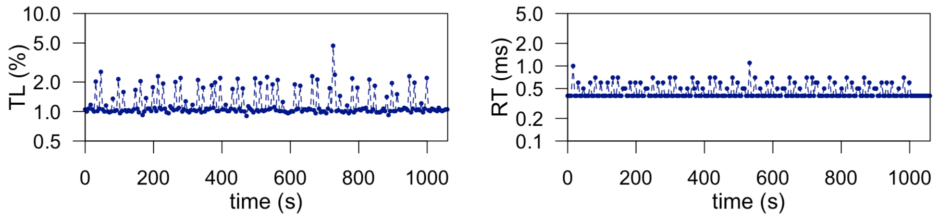
(a) TL (log scale) (b) RT (log scale)

Figure 10: Service metrics of the web server with L=10,000 *reqs/s* in attack-free conditions (adjusted configuration - *no reqtimeout*).

as (i) potential defense modules and (ii) configuration of the server –both assessed above– the *magnitude* of the load is another factor that might affect the service metrics. Experiments in Section 5.2 and 5.3 are done with a load L=1,000 *reqs/s*, which we keep intentionally low; nevertheless, according to the results in Figure 2, the server can handle safely much higher values of L. In this respect, it is worth investigating whether the attacks that seem "apparently" ineffective, such as CICIDS2017 hulk (Figure 9a) or ISCXIDS2012 HTTP DoS (Figure 9e), would instead disrupt the victim server when L is high. For all the following experiments, the server is operated with the *adjusted* configuration and *no* defense module in place.

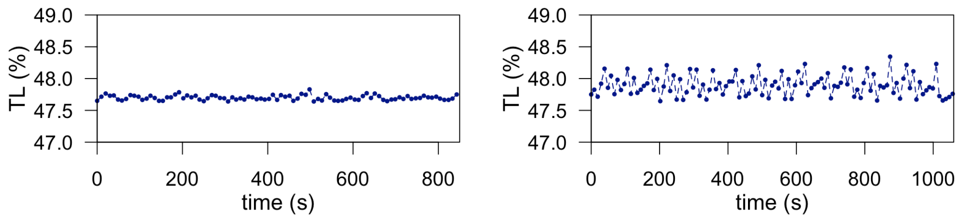
We monitor beforehand the service metrics in **attack-free conditions** with a load L=10,000 *reqs/s* –thus slightly above the *knee capacity*. This value of L is strongly relevant because it is the point where the throughput of the server stops growing linearly according to Figure 2a. Figure 10 shows TL and RT; the y-axis is given in log scale (and limited to the range [0.5, 10]% for TL) in order to appreciate the value of the metrics. We observe a mean TL of 1.05% within [1.02, 1.08] at 95% confidence; RT is 0.4 *ms* if not for sporadic spikes. It can be noted that the server suffers from a "natural" performance loss *irrespective from any concurrent attack*, which is intrinsically caused by the the high value of load.

On the other hand, Figure 11 shows the TL and RT obtained by replaying CICIDS2017 hulk while the server undergoes a "client" load L=10,000 *reqs/s*. The mean TL is 1.21% within [1.16, 1.27] at 95% confidence; similarly, the mean RT is 0.46 *ms* within [0.44, 0.47] at 95% confidence. Differently from Figure 9a –where TL=0% and L=1,000 *reqs/s*– CICIDS2017 hulk has now some effect on the server. Nevertheless, the effect of the attack consists



(a) TL (b) RT (log scale)

Figure 11: Service metrics of the web server with L=10,000 *reqs/s* in face of CICIDS2017 hulk (adjusted configuration - *no reqtimeout*).

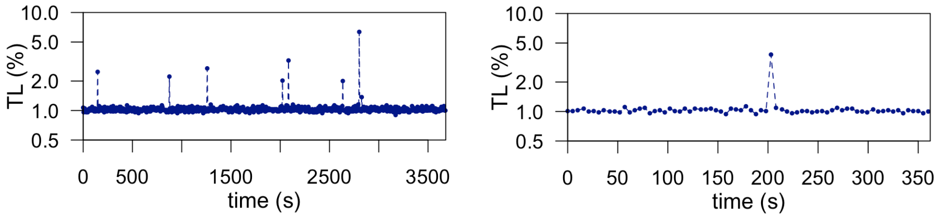


(a) Attack-free conditions (b) CICIDS2017 hulk

Figure 12: TL of the web server with L=20,000 *reqs/s* (adjusted configuration - *no reqtimeout*).

of negligible fluctuations over the *natural* performance loss of the server in attack-free conditions shown in Figure 10. Another interesting observation is that the RT caused by the attack is by far lower than the "typical" maximum tolerable delay for a response of a web server in order to be usefully deployed in many practical applications, such as multilayer workflows [44]. Similar considerations hold for much higher values of L. For example, Figure 12 (where y-axes are limited to the range [47, 49]% to appreciate the small variability of TL) shows the TL of the server in attack-free conditions and under CICIDS2017 hulk with L=20,000 *reqs/s*: the sample mean of the former series is 47.7%, while the latter is 47.9%. As in the previous case, most of the TL increase is explained by the load itself rather than by the attack, which sums marginal fluctuations over the attack-free baseline when the server is pushed beyond its knee capacity.

As for the results obtained by replaying the traffic from other public



(a) ISCXIDS2012 HTTP DoS (b) NDSec-1 2016 Syn flood #1

Figure 13: TL (log scale) of the web server with L=10,000 *reqs/s* in face of two attacks (adjusted configuration - *no reqtimeout*).

attacks, we observe very similar outcomes. Figure 13 shows the TL caused by ISCXIDS2012 HTTP DoS and NDSec-1 2016 SYN flood #1 under a "client" load L=10,000 *reqs/s*. Again, it can be noted that the mean TL is steadily around 1% (i.e., the attack-free value) if not for sporadic spikes.

Finding: A DoS attack *per se* explains only a part of the loss of the metrics measured for a victim server; as such, it is important to discriminate the actual impact of the attack from the underlying load. Surprisingly, for the public datasets assessed in this study we did not observe an "advantageous" interplay, i.e., more disruptive impact, between attack data and increasing load.

This finding is extremely relevant. Let us consider a practical example with CICIDS2017, where the `pcap` data file of the "DoS day" –July 5, 2017, used in our work– contains 1,486,069 total packets directed to the victim node over 8 *hours* of capture. The total breaks down into 1,383,651 packets originated by the attacker node and 102,418 from all the remaining sources; out of the latter contribution, only 128 packets are directed to the HTTP destination port 80 of the victim server. In consequence, it can be reasonably stated that the victim was serving almost *no* benign background activity, i.e., not related to attacks, at the time CICIDS2017 DoS data were collected. The lack of benign activity from legitimate clients *intertwined* with attack traffic, does not properly reflect the uncertainty of real-life operations.

6. Threats to Validity

As for any data-driven study, there may be concerns regarding the validity and generalizability of the results. We discuss them based on the four aspects of validity listed in [45].

Construct validity. Our analysis is based on five public datasets collected and made publicly available by *independent* research groups over the past years. The datasets consist of relevant categories of DoS attacks, which represent a major threat to the availability of real-life production servers. The study builds on experiments and measurements aiming to infer possibly-general insights about the representativeness of public data and attack traffic, which are widely used by the research community on intrusion detection systems. We are confident that the experimental details provided in the paper would support the replication of our study by future researchers and practitioners.

Internal validity. The results and key findings of this paper are based on both direct emulation and replay of third-party attack traffic. RELIVE has been validated with well-consolidated scripts and utility programs; experiments have been done with several attacks. We have taken into account different key factors, such as *defense*, *configuration* and *load*, to provide evidence of the actual impact of the attacks against the victim server. The use of such a mixture of diverse datasets and experimental conditions allows to mitigate strongly internal validity threats. Our experiments are founded on consolidated software tools and statistical indexes, such as the cross correlation function. The key findings of the study are consistent across the datasets and attacks, which provides the analysis with a reasonable level of confidence.

External validity. Our experiments can be applied to other similar intrusion datasets that provide network traffic and packet data files collected under DoS attacks. Given the wide spread of network capture programs and other tools for handling and transforming traffic formats, replicating and assessing the impact of previously-captured traffic data is definitively feasible in practice. In fact, in this paper we ported the experiments across five independent datasets, which mitigates external validity threats. The only source of overhead is the time needed to isolate DoS traffic out of wider network captures that are typically made available by the research groups. This time depends on the quality of the documentation of the datasets and the availability of other key details, such as source and destination of the attacks, start time and duration of the attacks.

Conclusion validity. Conclusions have been inferred by varying defense, configuration and load. Overall, these aspects are strongly overlooked by the literature. In this respect, it is worth noting that we *do not* develop around a unique configuration of the victim server; rather, we conduct repeated experiments in order to obtain a comprehensive picture of the service metrics under different experimental conditions. Comparisons have been made across the set of experiments to make sure that our findings have not been biased by a particular configuration of the victim server. The inferences made in the paper are consistent across the datasets. Our findings, which are strongly supported by data, contribute to establish new knowledge in the area and are strongly relevant to practitioners.

7. Conclusion

The issue considered in this paper stems from the observation that public intrusion datasets are widely used as benchmarks for intrusion detection algorithms and tools. Unfortunately, most of the times these datasets lack relevant information including data collection modalities, testbed configurations and, above all, impact of the attacks emulated. After a thorough examination of common datasets and of the accompanying information, it is almost natural to wonder if tens of proposals in the current intrusion detection literature have just been quite successful to detect attacks that are harmless for present-day hardware and software systems –being possibly useless with more dangerous ones.

In theory, a successful IDS should be able to detect any attack, whether it leads to the unavailability of the server or is relatively harmless. We do not support the position that only effective attacks should be detected, neglecting all the rest. However, it is a fact that often the network traffic generated under attack can substantially change depending on the server load and on the defense modules installed on the server [46]. Accordingly, IDSes may be unable to detect attacks generated in load conditions or server configuration different from the ones used for dataset collection.

In order to evaluate the representativeness of public intrusion datasets, i.e., to make clear at what extent these datasets can be useful for, we have reproduced in a controlled environment the DoS attacks in five publicly available datasets provided with network capture `pcap` files. This work has entailed the implementation of RELIVE, a traffic replay tool specially crafted to reproduce DoS attacks captured in the datasets. The analysis is limited to public datasets providing `pcap` data files and to attacks falling within the scope of RELIVE, which is addressed in Section 4.3. Whilst the current implementation is not a *one-fits-all* approach to replay any arbitrary attack, we address a significant class of DoS attacks, which keeps attracting substantial research efforts by the community. In brief, our findings show that sometimes the DoS attacks performed are harmless –often ineffective– against properly configured servers. The consequences can be drawn as follows:

- the production of an intrusion dataset is definitely a complex matter. It is not sufficient to set up a realistic testbed and to collect gigabytes of traffic produced by traffic generators and attack tools. The configuration of the web server in the case of DoS attacks has to be suitably hardened (who cares about weakened servers?). Hardware and software configurations have to be fully documented. The effect of the attacks carried out should be evaluated –it is not necessary to detect harmless attacks.

- the performance of intrusion detection algorithms and tools cannot be evaluated *solely* by the results obtained on present-day datasets. At least until a new generation of datasets following the principles set out above will be available, additional experimentation on realistic traffic and environments is indeed necessary to judge the validity of detection proposals.

In conclusion, our findings contribute to establish new knowledge in the area and pose novel open challenges. We hope that this effort will start a process leading to the construction of more rigorous security datasets. In the meantime, we are confident that the detection results obtained on currently available datasets are considered *cum grano salis*, avoiding to overlook the natural limits of partially-undocumented data collections.

References

[1] U. Franke, J. Brynielsson, Cyber situational awareness - A systematic review of the literature, Computers & Security 46 (2014) 18 – 31.

[2] D. Cotroneo, A. Paudice, A. Pecchia, Empirical analysis and validation of security alerts filtering techniques, IEEE Transactions on Dependable and Secure Computing 16 (5) (2019) 856–870.

[3] A. Sharma, Z. Kalbarczyk, J. Barlow, R. Iyer, Analysis of security data from a large computing organization, in: Proc. International Conference on Dependable Systems Networks, IEEE, 2011, pp. 506–517.

[4] E. Raftopoulos, X. Dimitropoulos, Understanding network forensics analysis in an operational environment, in: Proc. Security and Privacy Workshops, IEEE, 2013, pp. 111–118.

[5] C. Zhong, T. Lin, P. Liu, J. Yen, K. Chen, A cyber security data triage operation retrieval system, Computers & Security 76 (2018) 12 – 31.

[6] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho, A survey of network-based intrusion detection data sets, Computer & Security 86 (2019) 147–167.

[7] M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: Proc. Symposium on Computational Intelligence for Security and Defense Applications, IEEE, 2009, pp. 1–6.

[8] J. McHugh, Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, ACM Transactions on Information and System Security 3 (4) (2000) 262–294.

[9] G. Mantas, N. Stakhanova, H. Gonzalez, H. H. Jazi, A. A. Ghorbani, Application-layer denial of service attacks: Taxonomy and survey, International Journal of Information and Computer Security 7 (2/3/4) (2015) 216239.

[10] S. T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, IEEE Communications Surveys & Tutorials 15 (2013) 2046 – 2069.

[11] H. Liu, B. Lang, Machine learning and deep learning methods for intrusion detection systems: A survey, Applied Sciences 9 (20) (2019) 4396.

[12] M. Catillo, A. Pecchia, M. Rak, U. Villano, A case study on the representativeness of public DoS network traffic data for cybersecurity research, in: Proc. International Conference on Availability, Reliability and Security, ACM, 2020, pp. 1–10.

[13] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani., Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: Proc. International Conference on Information Systems Security and Privacy, SciTePress, 2018, pp. 108–116.

[14] W. Lee, S. J. Stolfo, A framework for constructing features and models for intrusion detection systems, ACM Transactions on Information and System Security 3 (4) (2000) 227261.

[15] M. Almseidin, M. Alzubi, S. Kovacs, M. Alkasassbeh, Evaluation of machine learning algorithms for intrusion detection system, in: Proc. International Symposium on Intelligent Systems and Informatics, IEEE, 2017, pp. 277–282.

[16] P. Kushwaha, H. Buckchash, B. Raman, Anomaly based intrusion detection using filter based feature selection on KDD-CUP 99, in: Proc. TENCON, IEEE, 2017, pp. 839–844.

[17] J. V. V. Silva, M. A. Lopez, D. M. F. Mattos, Attackers are not stealthy: Statistical analysis of the well-known and infamous KDD network security dataset, in: Proc. Conference on Cloud and Internet of Things, 2020, pp. 1–8.

[18] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, K. Das, The 1999 DARPA off-line intrusion detection evaluation, Computer Networks 34 (4) (2000) 579595.

[19] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, A. A. Ghorbani, Characterization of Tor traffic using time based features, in: Proc. International Conference on Information Systems Security and Privacy, 2017, pp. 253–262.

[20] N. Moustafa, J. Slay, UNSW-NB15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set), in: Proc. Military Communications and Information Systems Conference, IEEE, 2015, pp. 1–6.

[21] A. Shiravi, H. Shiravi, M. Tavallaee, A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Computers & Security 31 (2012) 357–374.

[22] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, R. Therón, UGR'16: A new dataset for the evaluation of cyclostationarity-based network idss, Computers & Security 73 (2017) 411 – 424.

[23] G. Loukas, G. ke, Protection against denial of service attacks: A survey, The Computer Journal 53 (7) (2010) 1020–1037.

[24] T. Mahjabin, Y. Xiao, G. Sun, W. Jiang, A survey of distributed denial-of-service attack, prevention, and mitigation techniques, International Journal of Distributed Sensor Networks 13 (12) (2017) 1–33.

[25] C. Douligeris, A. Mitrokotsa, DDoS attacks and defense mechanisms: a classification, in: Proc. International Symposium on Signal Processing and Information Technology, IEEE, 2003, pp. 190–193.

[26] S. alr, R. Atay, M. K. Pehlivanolu, N. Duru, Intrusion detection using machine learning and deep learning techniques, in: Proc. International Conference on Computer Science and Engineering (UBMK), 2019, pp. 656–660.

[27] F. Filho, F. Silveira, A. Junior, G. Vargas-Solar, L. Silveira, Smart detection: An online approach for DoS/DDoS attack detection using machine learning, Security and Communication Networks 2019 (2019) 1–15.

[28] X. Qu, L. Yang, K. Guo, L. Ma, T. Feng, S. Ren, M. Sun, Statistics-enhanced direct batch growth self-organizing mapping for efficient dos attack detection, IEEE Access 7 (2019) 78434–78441.

[29] S. Nguyen, V. Nguyen, J. Choi, K. Kim, Design and implementation of intrusion detection system using convolutional neural network for DoS detection, in: Proc. International Conference on Machine Learning and Soft Computing, 2018, p. 3438.

[30] L. Sacramento, I. Medeiros, J. Bota, M. Correia, FlowHacker: Detecting unknown network attacks in big traffic data using network flows, in: Proc. International Conference On Trust, Security And Privacy In Computing And Communications / International Conference On Big Data Science And Engineering, IEEE, 2018, pp. 567–572.

[31] M. Catillo, M. Rak, U. Villano, Discovery of DoS attacks by the ZED-IDS anomaly detector, Journal of High Speed Networks 25 (2019) 349–365.

[32] X. Peng, W. Huang, Z. Shi, Adversarial attack against DoS intrusion detection: An improved boundary-based method, in: Proc. International

Conference on Tools with Artificial Intelligence, IEEE, 2019, pp. 1288–1295.

[33] A. Botta, A. Dainotti, A. Pescapè, A tool for the generation of realistic network workload for emerging networking scenarios, Computer Networks 56 (15) (2012) 3531–3547.

[34] S. Hong, S. F. Wu, On interactive internet traffic replay, in: A. Valdes, D. Zamboni (Eds.), Recent Advances in Intrusion Detection, Springer, 2006, pp. 247–264.

[35] W. Feng, A. Goel, A. Bezzaz, W. Feng, J. Walpole, TCPivo: A high-performance packet replay engine, in: Proc. ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research, ACM, 2003, p. 5764.

[36] Y. Li, R. Miao, M. Alizadeh, M. Yu, DETER: Deterministic TCP replay for performance diagnosis, in: Proc. USENIX Symposium on Networked Systems Design and Implementation, USENIX Association, 2019, pp. 437–452.

[37] R. Jain, The Art of Computer Systems Performance Analysis, J. Wiley & Sons New York, 1991.

[38] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[39] S. M. Fontes, J. R. Hind, T. Narten, M. L. Stockton, Blended SYN cookies, US Patent 7,058,718 (Jun. 6 2006).

[40] J. L. Devore, K. N. Berk, Modern Mathematical Statistics with Applications, Springer-Verlag New York, 2012.

[41] F. Beer, U. Buehler, Feature selection for flow-based intrusion detection using rough set theory, in: Proc. International Conference on Networking, Sensing and Control, IEEE, 2017, pp. 617–624.

[42] T. Bowen, A. Poylisher, C. Serban, R. Chadha, C. Jason Chiang, L. M. Marvel, Enabling reproducible cyber research - Four labeled datasets, in: Proc. Military Communications Conference, IEEE, 2016, pp. 539–544.

[43] T. Lukaseder, L. Maile, B. Erb, F. Kargl, SDN-assisted network-based mitigation of slow ddos attacks, in: Proc. Security and Privacy in Communication Networks, Springer, Cham, 2018.

[44] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), ACM SIGCOMM Computer Communivation Review 40 (4) (2010) 63–74.

[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic, 2000.

[46] M. Catillo, A. Pecchia, U. Villano, Measurement-based analysis of a DoS defense module for an open source web server, in: V. Casola, A. De Benedictis, M. Rak (Eds.), Testing Software and Systems, Springer International Publishing, 2020, pp. 121–134.