Towards a Framework for Improving DoS Cybersecurity Experiments

Marta Catillo, Antonio Pecchia, and Umberto Villano

Dipartimento di Ingegneria
Università degli Studi del Sannio, Benevento, Italy
{marta.catillo,antonio.pecchia,villano}@unisannio.it

Abstract. In recent years, a number of solutions have been proposed for the detection of Denial of Service (DoS) attacks. Most of them have been tuned and tested by means of publicly available labelled datasets, which can be conveniently used to overcome the scarceness of real-life data gathered under incidents and attacks from production environments. Notwithstanding the value of existing algorithms in providing extremely high detection rates, there is little concern about the cybersecurity implications of the traffic data used, i.e., in terms of representativeness and impact on continuity and operations of the victim services.
This paper presents a starting step towards a framework for replaying and assessing DoS attacks. The framework aims to improve DoS cybersecurity experiments by allowing to replay previously-recorded attack traffic available in packets data files. It features a number of components, such as a victim and a load generator, that allow to conduct experiments in a controlled and configurable environment. Overall, this makes it possible to assess DoS traffic itself and contextualize the effect on the service under assessment and potential countermeasures. The framework is proven by means of direct DoS emulation and traffic replay.

Keywords: DoS · traffic replay · network capture dataset.

1 Introduction

Nowadays Denial of Service (DoS) attacks pose a significant threat on the availability of network services [10]. In this context, early attack detection and prevention are crucial in order to guarantee the continuity of service to the end user. Customarily, during a DoS attack a malicious user intentionally floods the target server with many service requests with the aim of clogging it or even interrupting its activity [12]. In such a scenario, the server is forced to allocate resources to process a multitude of requests so that it fails to provide services to other legitimate customers/users. In the last years, DoS attacks evolved into a *second generation*, the so-called Slow DoS attacks [16]. These use low-bandwidth approaches that exploit application-layer vulnerabilities. Thanks to the plethora of ready-to-use and easy-to-find attack tools available on the net, performing both flooding and slow DoS attacks is extremely simple and does not require any coding experience. However, if these tools have made extremely easy the task of attackers, on the other hand have led to the collection of datasets of network traffic under attack to be used for intrusion detection research.

In order to mitigate DoS attacks, a number of countermeasures have been taken over the years [17]. Some conventional approaches are based on the monitoring of the connection request rate. A requesting client whose connection request rate is higher than a pre-established threshold is marked as an attacker. However, such approaches have obvious weaknesses as in some cases a legitimate requesting user could have a short-term burst of connection requests without leading an attack. Moreover, they are mostly uneffective for Slow DoS attacks. In recent years, with the rapid diffusion of deep learning techniques, many machine learning-based DoS detectors have spread in the literature [9]. All these solutions achieve very encouraging results in terms of detection rate, which in some cases can even reach values close to 100%. In most cases, the experimentation and evaluation of a new detector takes place on public-domain datasets, which are used as a sort of benchmark to assess the validity of the proposal. These data are typically made available in the form of `pcap` trace files and correspond to several different attacks emulated in a test environment. Popular public-domain intrusion detection datasets are CICIDS2017 [13], UNSW-NB15 [11], ISCX 2012 [14]. Unfortunately, most of the times these datasets are used blindly, thus by overlooking the representativeness of the traffic data therein. As a matter of fact, in the relevant literature it is possible to observe the tendency to pay more attention to the tuning of the detection algorithm, than to the data on which these algorithms are evaluated. As this would invalidate the effectiveness of detection under real-world traffic conditions, it is necessary to ensure that research datasets conform to representative operating conditions.

This paper presents a starting step towards a framework for replaying and assessing DoS attacks. We aim to improve DoS cybersecurity experiments by allowing to replay previously-recorded attack traffic –available in packets data files– within a controlled and configurable environment. In particular, our tool leverages `pcap` trace files, which are typically generated by network utility programs such as `tcpdump`. Our framework has been designed to meet several open challenges in current DoS research:

– Public research datasets might not conform to representative operating conditions. Our replay tool could be a key component for assessing the impact of traffic data, with a consequent construction of more rigorous intrusion detection datasets. Noteworthy, many existing intrusion detection datasets come with `pcap` trace files that can be replayed within our framework;
– Many machine learning-based algorithms are currently spreading for the detection of network intrusions and, in particular, DoS attacks. Most of them are tested and evaluated with public domain datasets that contain DoS attacks launched in a simulated environment. Our framework can support in tuning detection algorithms by reliving the attacks within controllable and configurable victim services and load conditions;

– Finding the suitable "defense threshold" of the system under assessment is always a complex operation. Since our replay tool allows an effective evaluation of the impact of a DoS attack, it could be used to understand whether to strengthen or loosen the defenses as appropriate.

Although in the literature there are some traffic replay tools for network analysis purposes, there are no frameworks that allow to carry out traffic assessments in a structured way with the aim of conducting comprehensive security experiments. Our ultimate goal is to create a framework that collects a series of "best practices" –traffic replay is one of them– to conduct rigorous security experiments. The framework features a number of components, such a victim server and a load generator, that allow to replay attack traffic data in a controlled environment and to configure desired workload conditions. The framework is proven by means of direct DoS emulation and traffic replay with CICIDS2017, i.e., a recent dataset that is gaining massive attention by the community. Results indicate that our tool can reproduce the impact of a previously-recorded DoS and conveniently replay third-party data from an existing dataset.

This paper is organized as follows. Section 2 deals with related work. Section 3 describes our replay approach. The paper illustrates the framework in Section 4 and describes the experiments that aim to provide practical insights into its usage in Section 5. Finally, conclusions are drawn and our future work outlined.

2 Related Work

There are many solutions in the literature that propose detection algorithms tuned and tested by means of public domain intrusion detection datasets (trace-based). A machine learning-based DoS detection system is presented in [8]. The approach used by the Authors is based on inference and the detection rate achieved is 96%. In [7], instead, it is described a feature reduction method in order to detect DoS in a reduced feature space with the PART classifier. At best, the Authors hit a 99.98% recall for DoS Hulk. In [2] the Authors describe a hierarchical intrusion detection system that provides for the combination of several classifiers. In particular, following a hierarchical approach, the system uses three classifiers placed on different levels of the training phase. They reach an overall detection rate of 94.475% and a false alarm rate of 1.145%. Finally, in [4] the Authors propose a DoS anomaly detector that uses a deep autoencoder as core component of the infrastructure. The Authors highlight the potential of the proposal for 0-day attacks.

In this context of network security experiments, the use of appropriate tools for generating controllable, reproducible, and realistic network traffic is of extraordinary importance. Therefore, with the aim of testing environments for security products, over time several network traffic replay tools have spread. In general, replay tools can be either *stateless* or *stateful.* Those that follow a *stateless* approach replay packets according to their timestamps, and the content of replayed network packets is exactly the same as that stored in the original

Tool	*Is stateful?*	*Payload*	*Approach*	*Main feature*
`tcpreplay` [1]	No	Original payload	Trace-based	Replays traces at a specified rate
`tcpliveplay` [15]	Yes	Original payload	Trace-based	Replays traces using new TCP connections
`TCPOpera` [6]	Yes	Original payload	Statistical-based	TCP state emulation
`Our tool` [6]	Yes	Original payload	Trace-based	TCP replay for security experiments

Table 1: Replay tools - summary of comparison.

network traces. On the other hand, *stateful* replay tools are much more sophisticated. They manage the state of connections during replay, and therefore the content of replayed network packets may need to be altered in order to fit the "new" network configuration. As for the *payload* generation methods, instead, there are replay tools that do not alter the payload of the original packets, while others are able to replay packets with a new, "re-generated" payload. Another possible difference is between tools that follow a *trace-based* replay and others that perform a *statistical* replay. The former provide that the traffic sent over the network during replay is identical to the traffic contained in the capture file. The latter, instead, analyze the original captured traffic in order to collect statistical information (overall packet frequency, timing between packets, etc.) and generate new traffic traces that are similar to the original capture.

One of the most famous replay tools is surely `tcpreplay` [1]. In fact, it is a suite of tools containing a series of open-source UNIX utilities. In particular, `tcpreplay` it is a command-line tool designed to work with standard network cards. It simply replays previously captured traffic traces at a specified rate and does not actively modify the information of the transport layer header and the payload of a packet. However, `tcpreplay` is completely stateless and is unable to handle the update of TCP sequence and acknowledgement numbers. Therefore, it does not support replaying data exchanges to/from a server. In order to overcome this limitation, the same suite provides `tcpliveplay` [15], a tool that replays packet captures statefully by keeping track and updating TCP sequence numbers. Finally, a tool that performs a statistical replay is `TCPOpera` [6]. It is designed for a stateful emulation of TCP connections. `TCPOPera` does not provide trace-based replay of the captured data. It first develops analytics from a packet trace, then creates a statistical model of the identified events, and finally generates synthetic traffic flows from the model.

The tool on which hinges the framework proposed in this paper uses a stateful replay approach, starting from the captured DoS traces in `pcap` format and preserving the original payload of packets. Unlike the aforementioned tools, designed essentially for network analysis or diagnostic activities, it is conceived as a lightweight, ready-to-use solution for replaying and assessing DoS attacks that typically appear in `pcap` format in most intrusion detection datasets. It is, therefore, a key tool for conducting a robust experimentation and validation of the detection techniques commonly tested and tuned on public DoS network traffic data for cybersecurity research. Table 1 summarizes the characteristics of the above cited replay tools with respect to our proposal.

3 Ongoing Replay Tool

3.1 Design

As briefly mentioned in the introduction, our framework hinges on a tool that allows to replay previously-recorded network traffic under realistic conditions, so as to measure the impact of attacks on a suitably-configured web server and load conditions. One of the hallmarks of our replay tool is surely simplicity, both in terms of design and use. As a matter of fact, given an input `pcap` file, it can be used as a common command line tool. In particular, the input `pcap` trace file contains the network capture we want to replay (that in our case includes a DoS attack to a web server). `PCAP` (Packet CAPtures) files are commonly used for storing traffic network traces. Currently there are multiple widely accepted `pcap` formats, but one of the most popular is `LibPCAP` [5]. The name derives from the library of the same name, where it is formally defined. `LibPCAP` is the oldest `pcap` format, but it is the most popular, mainly because is the default format used by widely used network applications such as `tcpdump`. In addition, most intrusion detection datasets provide DoS traces in `LibPCAP` format. The typical structure of a `LibPCAP` file is shown in Figure 1. In particular, the first element is a *Global Header (GH)* with general traffic information, such as the correction to UTC time or the specific endianness (big/little). There then follow packets characterized by *Packet Header (PH)* (including timestamps and data sizes) and *Packet Data (PD)*.

As previously mentioned our replay tool follows a *trace-based* replay approach. In particular, its ultimate goal is to relive previously-recorded traffic towards arbitrary IP addresses and over brand-new sockets and connections, without altering the payload of the original packets. Therefore, since the tool supports replaying data exchanges to/from a server, it can be considered fully

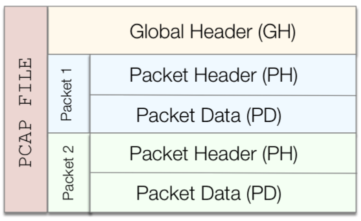

Fig. 1: LibPCAP file structure.

```
09:07:05.766440 IP 192.168.56.102.39842 > 192.168.56.101.http:
 Flags [S], seq 2633020550, win 64240, options [mss 1460,sackOK,
 TS val 2720361938 ecr 0,nop,wscale 7], length 0
09:07:05.766674 IP 192.168.56.102.39842 > 192.168.56.101.http:
 Flags [.], ack 1353780858, win 502, options [nop,nop,TS val
 2720361938 ecr 153334785], length 0
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 [nop,nop,TS val
 2720361939 ecr 153334785], length 19: HTTP: GET /?12 HTTP/1.1
```

Fig. 2: Human-readable `tcpdump -r` of three example packets.

stateful. This feature is not trivial, as most existing replay tools are not able to replay traces using new TCP connections. It is worth pointing out that the proposal closest to our work, in that offers a similar replay service, is `tcpliveplay` [15]. Unfortunately, the use of `tcpliveplay` for our purposes has not produced the expected result. The weaknesses of `tcpliveplay` are also confirmed by the intense ongoing bug fixing activity carried out by the community[1].

Unlike other available software, our tool replays traffic *at the same rate* it was originally recorded in the `pcap` file. This functionality is extremely important to reproduce effectively the original traffic conditions. The tool has been implemented in `python`, exploiting `dkpt3`[2], i.e., a module for fast packet creation and parsing with definitions for the basic TCP/IP protocols.

3.2 Approach

The whole replay process, which starts from the acquisition of a DoS `pcap` file and leads to the replay of the original trace, is described below. From the input `pcap`, containing a previously-recorded DoS attack, we extract the packets sent by a given source address to a given destination. More specifically, source and destination identify the attacker and the victim, respectively. A visual and simplified representation of the `pcap` content is shown in Figure 2. It shows a human-readable `tcpdump -r` of three example packets (one packet spans three lines) sent by `IP 192.168.56.102` to `IP 192.168.56.101`. These addresses match the IP of the attacking and the victim node used in our testbed, presented in the next section. It is important to note that the figure shows just a small window of the input `pcap` file, which typically consists of millions packets.

Given the input packets, since we want to replay a two-way communication with data exchanges to/from a server, the destination address is *rewritten* with the address of the server node towards which the traffic is intended to be sent for replay purposes. Then the adjusted file is scanned sequentially. For each packet in the file, the tool either (i) discards it (at this stage of development. our focus

[1] https://github.com/appneta/tcpreplay/issues/540
[2] https://dpkt.readthedocs.io/en/latest/

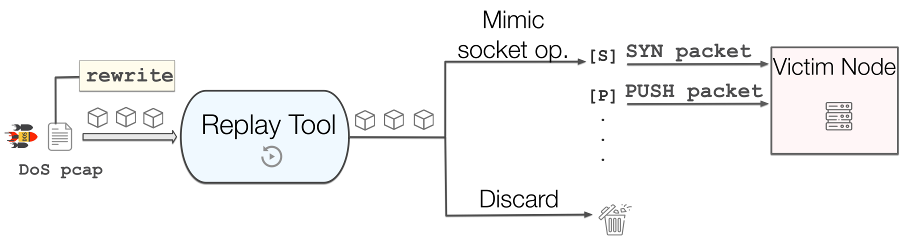

Fig. 3: Replay process.

is only on TCP traffic, and everything else is discarded) or (ii) mimics socket operations based on the value of the *Flags* field, shown at the rightmost part of the lines in Figure 3. The *Flags* letter is meaningful for the operation to be performed. For example, the tool initiates a new socket connection upon `[S]` (i.e., a `SYN` packet) or sends data upon `[P]` (i.e., a `PUSH` packet). The operations are differentiated on the basis of the timestamp of the packets contained in the original capture. This allows to intercept and replay the actual timing of the recorded packets. Moreover, at any time the tool maintains a suitable number of active concurrent connections towards the destination address, based on the actual number of socket opening and closing operations encountered across the input packets. The whole replay process is depicted in Figure 3.

It is worth noting that the current implementation of the replay tool is not "tuned" to manage some isolated cases. For example, in the original `pcap` trace there may be packets corresponding to `GET` URL requests valid on the "original" victim node, but invalid for the server on which the replay experiment is being performed. Furthermore, the tool might accidentally attempt to replay SSH traffic with consequent and unavoidable authentication issues. Although we have not yet addressed these issues, the current implementation allows to successfully replay a significant class of DoS attacks. In particular, we are able to replay both flooding and slow DoS. The two DoS categories exploit distinct attack strategies, and therefore they leave different "fingerprints" inside the trace files. The behavior of the replay tool is always aligned to the specific type of attack to be replayed (more on this later).

4 Proposed Framework

We are striving for a comprehensive framework consisting of a number of components that allow to conduct controlled and configurable DoS experiments. Whilst our replay tool is a key addition towards addressing several open challenges in DoS research stated in the Introduction, a typical security experiment includes further steps, such as attack emulation, traffic data collection and evaluation of performance metrics. As such, our "core" traffic replay tool is meant to be instantiated in a controlled network testbed featuring a *victim sever*, i.e., the target of the assessment, a *load generator*, i.e., the component supplying a configurable workload to exercise the victim, and a supplementary *attacker node*, which allows to conduct controlled attack emulations aimed at complementing the findings obtained by means of replay. Although not covered in this starting stage, in the future the framework will be enriched with additional facilities, such as monitoring dashboards, network flows extractors and intrusion detectors. Overall, the framework is conceived to follow a series of "best practices" in computer performance evaluation.

4.1 Nodes and Usage Modes

In this current proposition, the framework above is instantiated by means of four network nodes on a LAN, as follows:

- `Attacker_1:` emulates the attack by means of a suitable DoS tool hosted on a Kali Linux node.
- `Attacker_2:` allows to replay the DoS attack by reliving the network traffic gathered from a DoS previous capture stored in a `pcap` file.
- `Victim:` Apache 2.2 web server node.
- `Load generator:` triggers, through `httperf`[3], HTTP requests that serve as benign background load to exercise and to check the status of the web server during the experiments.

Further information about the nodes is reported in Table 2. Noteworthy, the framework we set up allows us to make experiments that involve both direct emulation and replay of previously-recorded attacks. As such, it supports two distinct usage modes:

1. *attack emulation mode:* consists in emulating real attacks by means of a given DoS tool;

[3] https://github.com/httperf/httperf

IP address (role)	*Operating System*	*Application*
`192.168.56.101` (Victim)	Ubuntu LTS 12.04	Apache 2.2 web server
`192.168.56.103` (Load Generator)	Ubuntu LTS 12.04	httperf
`192.168.56.102` (Attacker_1)	Kali GNU/Linux Roling 2019.4	DoS tool
`192.168.56.1` (Attacker_2)	macOS Sierra 101.2.6	proposed replay tool

Table 2: Nodes description.

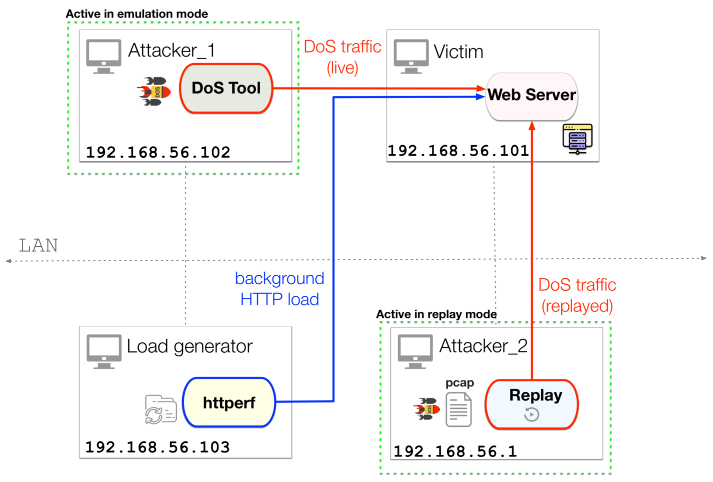

Fig. 4: Proposed assessment framework.

2. *attack replay mode:* consists in replaying a DoS attack by reliving the network traffic from a previous capture.

Attacker nodes are used as follows. `Attacker_1` allows emulating real attacks by means of various DoS tools hosted by the Kali Linux node: it is unused in *replay* mode. `Attacker_2`, instead, allows to replay an attack by reliving the network traffic from a previous capture stored in a `pcap` file: it is unused in *emulation* mode. Therefore, during the experimentation phase, the two attackers are never simultaneously active. It is worth noting that in both operating modes the web server is exercised concurrently with both DoS traffic (either emulated or replayed in the two modes, respectively) and benign background HTTP load generated by `httperf`. We selected the Apache web server as a significant case study, given its widespread use. In Fig. 4 we show a complete representation of our framework and its components.

4.2 Configurations and Evaluation Metrics

In order to emulate a realistic experiment, we adjusted the web server default parameters (that can be found at `/etc/apache2/apache2.conf` in a typical Linux-based system) because they may not be representative of real-life production server. In the adjusted configuration we set `MaxKeeplAliveRequests`, i.e., number of requests over the same connection, to 0 (unlimited), `ThreadLimit` to 1024, `ThreadPerChild` to 256 and `MaxClients` to 20148. These are crucial parameters for making realistic assessments on production servers.

Our current focus is on DoS attacks. This type of attack often involves the opening of a substantial number of concurrent connections to the target. By default, Unix-like machines often have a ceiling to the allowed number of simultaneously open files (1024 by default) and hence to the maximum number of currently opened sockets available to a process. We set this number to 200,000 with `ulimit -n` before launching the attack. This number is large enough to avoid any interference or saturation effect by the operating system. As a further note, we conducted a capacity analysis of the web server before performing any experiment, in order to discover the maximum load that can be handled by the server in attack-free conditions. The details of the capacity analysis are omitted here due to space limitations; however, we found out that in normal DoS-free conditions the RT of the server in our testbed is 0.2 *ms*, without exceeding its maximum load capacity.

The selected metrics for the evaluation of our experiments are as follows.

- **Load (L):** the desired level of load to stress the web server during a testing timeframe. We set this value by exploiting parameters supported by `httperf`, such as *total connections*, *HTTP requests per connection* and *connection rate*. The load submitted to the server is measured in HTTP requests per second (*req/s* in the following).
- **Response Time (RT):** the time taken to serve a request measured in milliseconds (*ms*). It is useful for evaluating server performance. For completed HTTP requests, the response time is a fundamental index as it impacts the QoS perceived by the application users.

5 Experimental Results

Experiments aim to provide practical insights into the usage of the proposed framework to replay DoS attacks. Evaluation is **twofold**: (i) replay of a fully "controlled" DoS attack generated within our framework, and (ii) replay of a DoS taken from a state-of-the-art public research dataset. The former is intended to demonstrate the ability of our framework at reproducing the impact of a DoS; the latter entails a potential use case of the framework in assessing the resiliency of a victim service by leveraging existing third-party malicious traffic. The experiments presented here focus on **Slowloris**, which is a DoS attack that aims to saturate the victim server, opening connections, but never completing the HTTP requests. It is a well-known *application-level* attack, often effective: as such, it is strongly relevant in the context of our work.

5.1 Replay of a Controlled DoS

We mimic a real Slowloris DoS, beforehand. Emulation is done in our controlled network by attacking the victim server, i.e., `192.168.56.101`, with a well-consolidated Slowloris GitHub tool[4] hosted by the Kali Linux node, i.e., `Attacker_1`. This will be referred to as the "original" attack throughout this Section. Moreover, during the progression of the attack the victim undergoes a concurrent, benign, load of 1,000 *reqs/s* generated by `httperf`, whose response time is monitored in order to assess the impact of the DoS. It should be noted

[4] `https://github.com/gkbrk/slowloris`

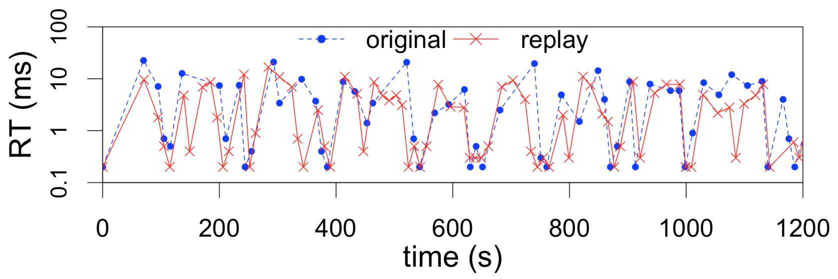

Fig. 5: RT during the original attack and its replay (y-axis is in log scale).

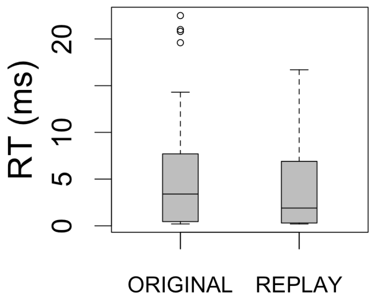

Fig. 6: Boxplot of the response time.

that at this stage we also record the attack traffic in a `pcap` packet data file, so that it can be replayed later on, which is the purpose of our framework.

Fig. 5 (●-*marked* series) shows how RT varies during the progression of the original attack. In the plot, the x-axis represents the time since the beginning of the experiment; the y-axis is given in log scale to better appreciate fluctuations around low RT values. The duration of the experiment is large enough to collect a large sample, i.e., $\gg$30 observations, in order to make statistical claims. The attack impacts service operations, whose RT = 0.2 *ms* in attack-free conditions, and –occasionally– it causes up to RT=10 *ms*, which is a significant degradation. Noteworthy, 10 *ms* is seen as a typical maximum *tolerable* delay for a response of a web server in order to be usefully deployed in multilayer workflows [3].

The second part of the experiment consists in replaying the traffic recorded during the "original" attack: our aim is to verify whether we can reproduce similar effects on the server. As in the previous experiment, the victim server is exercised with a load L=1,000 *reqs/s* during the progression of the replay; however, the attacker is now represented by the **replay node** (which is fed with the `pcap` file recorded) rather than Kali Linux. RT measured during the replay is shown in Fig. 5 ($\times$-*marked* series) and superimposed to the original attack for better comparison. Interestingly, the "appearance" of the time series is similar: replay seems to reasonably well mimic the original attack.

Table 3: Summary of RT statistics within the original attack and its replay.

	mean	**standard deviation**	**95% confidence interval**
ORIGINAL	3.97	4.02	(2.96, 4.99)
REPLAY	3.40	3.77	(2.59, 4.22)

Beside the *visual test*, we conduct further statistical analysis. Fig. 6 shows the paired boxplots of the response time of the original attack and its replay. The leftmost boxplot highlights few sporadic outliers (depicted by circles) in case of the original attack. Most notably, IQRs[5] (Inter Quartile Ranges), which catch the dispersion of the RT around the median value, strongly overlap. Table 3 shows sample mean, standard deviation and 95% confidence interval of the RT observations after filtering out the outliers. It can be noted that the sample mean of RT during the original attack, i.e., 3.97 *ms* is within the CI of the replay and viceversa: as such, it can be reasonably claimed that the impact of the original attack and its replay are not statistically different. For the data in hand, the proposed replay framework produces an attack that is *statistically* the same as the original in terms of the effects on the server.

5.2 Replay of a Third-party DoS

We present here a replay experiment done with the Slowloris attack traffic of CICIDS2017. It is worth noting that CICIDS2017 is a recent dataset that gained massive attention by the community: the experiment is done to show the potential of our framework at handling *unseen* third-party traffic. Given the `pcap` packet data file available at the dataset's webpage[6] we (i) extract all the packets sent by `172.16.0.1` to `192.168.10.50` (i.e., attacker and victim in CICIDS2017, respectively) within the timeframe of the Slowloris attack emulated by the Authors and (ii) rewrite the destination as `192.168.56.101`, which is the address of the victim server in our testbed. The `pcap` obtained is fed to the replay framework and run against our victim web server. Such as the previous experiments, the victim undergoes a concurrent, benign, load of 1,000 *reqs/s*, which is is monitored to measure RT.

Fig. 7 shows how RT varies during the replay of CICIDS2017 Slowloris; again, the y-axis is in log scale. Surprisingly, it can be noted that the attack causes only marginal fluctuations above the normative response time of the server, which is 0.2 *ms* in attack-free conditions. Another interesting outcome is that RT under attack is always significantly lower than 10 *ms* (again, the indication of the maximum tolerable delay for a response in many practical settings), which is a further

[5] The Inter Quartile Range (IQR) of a boxplot is the difference between the *third* and *first* quartile.
[6] `https://www.unb.ca/cic/datasets/ids-2017.html`

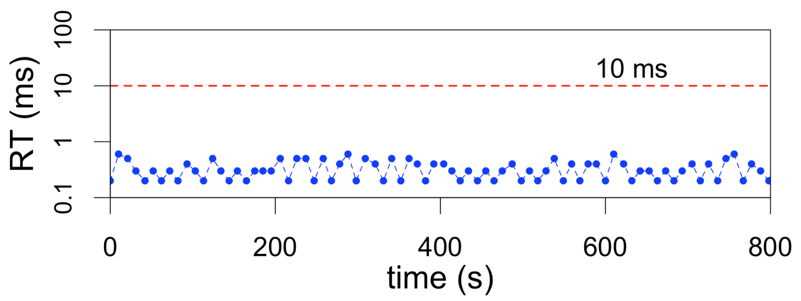

Fig. 7: RT during CICIDS2017 Slowloris (y-axis is in log scale).

remark of the scarse effectiveness of the attack. This is quite surprising given the large body of literature on anomaly detection that capitalizes on CICIDS2017. A closer look into CICIDS2017 Slowloris traffic revealed that –although somewhat relevant due to the abusive consumption of network resources– the attack was not disruptive enough against a well "tuned-up" server resembling a real-life configuration, such as the one of our testbed.

Although beyond the scope of this paper, this finding has major practical implications when it comes to the representativeness of public datasets for cybersecurity research and it will be investigated in the future. Noteworthy, it provides some initial insights into the above-mentioned challenges in DoS research, which pertains to the potential limitations of existing dataset in conform to representative operating conditions.

6 Conclusions

The goal of our current work is to meet several open challenges in DoS cybersecurity experiments, with a focus on the quality of network capture datasets. The paper is a step towards the design of a framework for replaying DoS attacks and the implementation of a prototype tool that allows to relive previously-recorded network traffic.

Our work is driven by the observation that research datasets might not conform to representative operating conditions. We show the validity of our proposal by means of a DoS attack replay experiment. This is meant to evaluate both the replay of a fully controlled attack and the replay of a DoS taken from a state-of-the-art public research dataset. The results show that the proposed framework replays a DoS attack obtaining statistically the same effects on the server as the original one. The use of the framework to replay of a DoS from a state-of-the-art dataset will give insight on the actual representativeness of public datasets widely used for cybersecurity research. This analysis will be conducted in the future.

The ultimate goal of our work is to develop an integrated set of tools that enable traffic assessments in a structured way, with the aim of creating complete security experiments. In particular, we aim to create a *ready-to-use* framework for both researchers and practitioners that enables rigorous security experiments starting with the deep analysis of the data and ending with the tuning of the detection algorithm.

Acknowledgments

Removed for double-blind review reasons

References

1. Aaron, T., Bing, M.: Tcpreplay tool (2012), `https://tcpreplay.appneta.com`
2. Ahmim, A., Maglaras, L.A., Ferrag, M.A., Derdour, M., Janicke, H.: A novel hierarchical intrusion detection system based on decision tree and rules-based models. In: 15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019. pp. 228–233. IEEE (2019)
3. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M.: Data center tcp (dctcp). In: SIGCOMM Comput. Commun. Rev. vol. 40, pp. 63–74. ACM (2010)
4. Catillo, M., Rak, M., Villano, U.: Discovery of DoS attacks by the ZED-IDS anomaly detector. Journal of High Speed Networks **25**, 349–365 (2019)
5. Harris, G.: Development/libpcapfileformat (mar 2011), `https://wiki.wireshark.org/Development/LibpcapFileFormat/`
6. Hong, S.S., Wu, S.F.: On interactive internet traffic replay. In: Valdes, A., Zamboni, D. (eds.) Recent Advances in Intrusion Detection. pp. 247–264. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
7. Kshirsagar, D., Kumar, S.: Identifying reduced features based on ig-threshold for dos attack detection using part. In: Hung, D.V., D´Souza, M. (eds.) Distributed Computing and Internet Technology. pp. 411–419. Springer International Publishing, Cham (2020)
8. de Lima Filho, F.S., Silveira, F.A.F., de Medeiros Brito Júnior, A., Vargas-Solar, G., Silveira, L.F.: Smart detection: An online approach for dos/ddos attack detection using machine learning. Security and Communication Networks **2019**, 1574749:1–1574749:15 (2019)
9. Liu, H., Lang, B.: Machine learning and deep learning methods for intrusion detection systems: A survey. Applied Sciences **9**(20), 4396 (Oct 2019)
10. Mantas, G., Stakhanova, N., Gonzalez, H., Jazi, H., Ghorbani, A.: Application-layer denial of service attacks: Taxonomy and survey. International Journal of Information and Computer Security **7**, 216 (01 2015)
11. Moustafa, N., Slay, J.: Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw- nb15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS) pp. 1–6 (2015)
12. Purwanto, Y., Kuspriyanto, Hendrawan, Rahardjo, B.: Traffic anomaly detection in ddos flooding attack. In: 2014 8th International Conference on Telecommunication Systems Services and Applications (TSSA). pp. 1–6 (2014)
13. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,. pp. 108–116. INSTICC, SciTePress (2018)

14. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security **31**, 357–374 (05 2012)
15. Siam, Y.: Tcpreplay tool (dec 2013), `https://tcpreplay.appneta.com/wiki/tcpliveplay-man.html`
16. Sikora, M., Gerlich, T., Malina, L.: On detection and mitigation of slow rate denial of service attacks. In: 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). pp. 1–5 (2019)
17. Taghavi Zargar, S., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. IEEE Communications Surveys &amp Tutorials **15**, 2046 – 2069 (11 2013)