An extensible evaluation system for DoS research

Nik Sultana
University of Pennsylvania

Shilpi Bose
University of Pennsylvania

Boon Thau Loo
University of Pennsylvania

Abstract

Denial-of-Service (DoS) is a common form of cyber-attack, and application-layer DoS is an increasingly common form of DoS that exploits the behaviour of applications on the network. Running application-layer DoS experiments is tedious because of the configuration and dependency-management needed. Currently there is poor support for researchers to run such experiments on their clusters, and this in turn hinders the development of DoS mitigations.

This paper describes DoSarray, a tool for running application-layer DoS experiments on a computer cluster by using containers to both simplify the experiment setup and to create networks containing $\times 10$ to $\times 100$ larger address diversity than physical hosts. This tool is also intended to complement prior work that automates volumetric DoS experiments on research testbeds.

DoSarray manages most mundane features of experiments: ranging from provisioning and pre-experiment checks, to measurement, analysis and graphing. We make the tool available for others to use and extend.

1 Introduction

This paper sets out to answer the question: How do we run application-layer DoS experiments in our cluster? We asked ourselves this question while researching DoS attacks and mitigations, and despite the availability of DoS attack *scripts* on the Internet, there lacks a system to use these scripts as part of an *experiment*.

There is much more to running a computational experiment than simply running a single computer program. As with any experiment, a computational experiment requires meticulous care to record the process used to set up and execute the experiment, gather data and interpret it. For computational experiments, this includes version numbers of software and its dependencies, input parameters, and configuration.

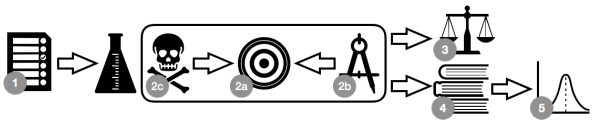


Figure 1: Steps taken when carrying out an experiment, detailed further in §4.2. (1) The network and virtual hosts are configured; (2) the experiment is carried out by (a) starting the target, (b) starting measurements of availability, and (c) running the attack(s); (3) load is polled to assess whether the results might have been skewed; (4) measurement logs are gathered and (5) analysed and graphed.

Further, DoS experiments necessitate generating attacks of sufficient *scale* in order evaluate how the attacks put pressure on systems and defenses. We mean scale not in the sense of traffic volumes *alone*, but also in the sense of *address diversity* to simulate larger networks: one needs to simulate attacks coming from (or proxied through) a larger number of hosts than are normally physically available on an academic research cluster. Large attack diversity is necessary to test mitigations and evaluate how they respond to larger attacks. A larger number of attackers or proxies puts pressure on the method that the mitigation uses to record state about hosts on the network.

In addition to automating the measurement of a DoS attack's effect and running larger experiments, there are two additional important properties of the computational experiments we seek to make. The first is *fidelity*: we prefer to run the actual attacking program, rather than replay a short packet sequence to saturate a 10Gbps link for example. The second property is *reproducibility*. In addition to being useful for other researchers, reproducibility is very useful to the original researchers since an experiment is likely to be run several times, and with varied parameters during exploratory periods when trying out new targets, attacks or mitigations.

In this paper we describe the DoSarray system that targets the scenario described above for running DoS experiments on computer clusters. DoSarray consists of a set of tools that are used in a workflow. The workflow is sketched in Fig. 1, and detailed in §4.2. A running example of using this workflow is given in §3.

There exists a wealth of research on DoS experiments [15] and automating them [14], and that work mostly concentrates on volumetric attacks and relies on a shared research testbed such as DETERLab [11]. By comparison, the work in this paper facilitates running experiments in one's own cluster, though there is no reason why DoSarray would not run on testbeds such as DETER. Running experiments in a cluster facilitates development and experimentation, which would be more difficult in a shared testbed.

Contributions and Paper Outline.

- We describe the design and implementation of a system for setting up DoS experiments (§4) to assess the success of the attack (and conversely, its mitigation). We evaluate the system's effectiveness, experiment fidelity, and extensibility (§5).
- We develop a novel visualisation, using contour maps, to compactly show the effect of a DoS attack on availability and latency (§4.3.1).
- We use DoSarray to carry out a broad evaluation of various HTTP services, including web servers and proxies (§6).
- The DoSarray scripts, and the data for the running example used in this paper are available online (§9).

We next outline the problem's background and related work (§2), after which we give an example use of DoSarray (§3).

2 Background and Related Work

The most closely-related work is by Mirkovic et al. [15, 13, 12, 17], who carry out a comprehensive study of DoS mostly based on experiments on the DETERLab [11] testbed. Their research ranges from how best to measure the effects of DoS, to how to automate experiments for the convenience of researchers.

DoSarray is complementary to testbeds such as DETERLab, on which it could be run. Using testbeds for development is difficult since the testbed needs to be shared with other users, and the fluctuating availability of testbed nodes might slow development. Thus our setup needs to work on local clusters, on which development is done, before test and evaluation on a larger testbed.

Other systems for cybersecurity experiments include NRL CORE [3] and CyberVAN [6], both of which rely on simulation, and offer sophisticated features to model various network features, particularly for ad hoc networks. DoSarray is much simpler in comparison: it does not attempt to model specific network devices and properties.

In addition to research tools, we also came across commercial "stressor" services, that can launch a DoS attack against a target for a fee, supposedly to test the target's resilience. We kept well away from these services since they are often run by cyber-criminals [2, 1].

There is ample prior work on using virtualisation for reproducible research [8, 7]. In recent years reproducibility has become a much more valued quality in research [9], spurred by the difficulty of replicating experiments done by others [4, 5]. DoSarray complements this line of work by offering a design and tool to help researchers make their research reproducible for themselves and others.

3 Example: Slowloris

In this section we describe a typical experiment scenario that will serve as a running example in later sections. In this experiment we test the responsiveness of the Apache web server when attacked with Slowloris, an HTTP-level attack that has two effects: (i) it stalls when sending HTTP headers to the server, and (ii) it can also flood the server with a large number of connections.

Experiment timeline. After 10 seconds of running Apache, we attack it from a single host using Slowloris for 20 seconds. We then let Apache run for 30 more seconds, to watch its recovery, before stopping the experiment.

Setup details. We use Apache version 2.4.26 in default configuration, and the de facto Slowloris Perl script obtained from the Internet. We measure availability using httping (§4.2.2), and carry out the experiment in a network of size 281: one host running Apache, another host running Slowloris, and the remaining hosts measuring Apache's availability throughout the experiment.

Results. Fig. 2 shows that the attack succeeds. The measurement nodes simulate bona fide clients which are trying to obtain a service from Apache; the DoS attack succeeds if Apache's availability is diminished from the viewpoint of the measurement nodes. The measurement nodes log their observations, and these logs are processed to produce various forms of output. Other outputs are described in more detail in §4.3.

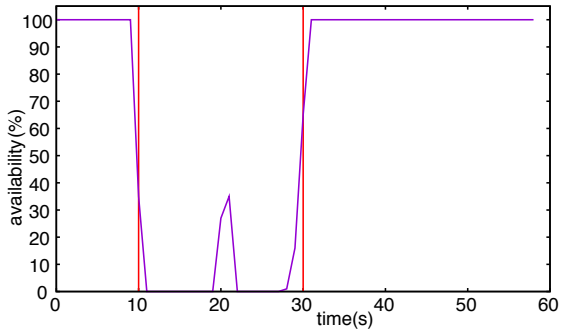


Figure 2: Availability over time, from our running example (§3). "Availability" here consists of the percentage of measurement nodes who successfully sampled the target during that point in time: 100% means that the target is responsive to all measurement nodes, whereas at 0% the target appears available to none. The vertical red lines indicate the attack window. The bump in the middle indicates that some measurement nodes briefly managed to get a response from the target during the attack.

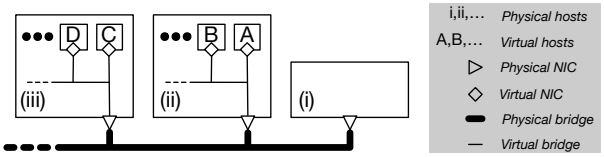


Figure 3: Bridge between physical and virtual hosts. We run the target (e.g., Apache) directly on physical hosts and use the virtual hosts for everything else.

4 DoSarray

This section expands on Fig. 1 to describe the design and use of DoSarray.

4.1 Setup and configuration

We set up a network of hosts on which to run the experiment. We use two kinds of hosts – physical and virtual – on which we run different kinds of software.

Fig. 3 shows our network setup. For virtual hosts we use Docker instances since they incur less overhead than VMs by default [10]. We bridge the physical and virtual networks (which are not NAT'd by the physical host) and set up static routes between hosts using standard GNU/Linux tools. Another benefit from using Docker containers is the ease with which to organise the scripts, tools, and dependencies, to recreate the exact environment across different machines.

The hosts run four kinds of programs:

1) Target. This is the software being attacked – such as Web or DNS servers.

2) Measurement. These are programs that simulate users trying to use the target, and their output is used to determine the success of the attack: if measurement programs cannot access the target (or the quality of access is significantly reduced) then the attack is succeeding. The measurement data is used to generate graphs such as Fig. 2.

3a) Attackers. Even the simplest attack scripts can be used to generate attacks of different intensities, durations, and combinations. This is described further in §4.2.3.

3b) Proxies. These get used by attackers, for instance to amplify the attack on the server. Not all attacks feature proxies.

A physical host is reserved to run the target exclusively (i.e., no virtual hosts) since we want it to use as many machine resources as it needs. In comparison, measurement and application-layer attack (and proxy) scripts usually require less resources, so we run them in virtual hosts.

In the example (§3) we had 8 physical hosts: we ran Apache on a physical machine, and used the remaining machines to instantiate 40 virtual hosts each. One of the virtual hosts was used to run Slowloris, and the remaining 279 ran the measurement. We can easily configure DoSarray to have multiple attackers in our experiment, by adding node names to a list. We can also have different kinds of attacks happen simultaneously, by editing a script.

4.2 Experiment protocol

We now describe each step of the workflow shown in Fig. 1. We tend to follow the same check-list for each experiment, but vary the experiment components (such as the attacker or target) and parameters (such as those given to the attack scripts, or to the measurement scripts).

4.2.1 Preparation and setup

After having started all the physical machines, we start the target on one of them, and start Docker on the rest. We create and start the Docker instances on each physical machine. We ensure that networking is set up and working as intended – each virtual instance should have a distinct IP address that is not NAT'd by the physical host.

At this point we have completed Step 1 in Fig. 1. We can then start the experiment. Experiments begin by starting the target (step 2a in Fig. 1) followed by the measurement scripts (step 2b). Our choice of measurement is described next.

4.2.2 Measurement

For the entire duration of the experiment, we measure. Measurements fall into two categories: *samples* and *load-tests*. Sampling measurements (e.g., those made using httping) consist of regular requests made to the target, to observe its response and the response's latency. Load measurements (e.g., using Apache Bench) consist of bursts of requests made to the target, to observe how it copes under that load.

Sampling measurements are better suited to our setup because they provide a stream of indications of the target's *availability*, which is the quality by which we determine whether a DoS attack succeeds. Measurement nodes play the role of bona fide clients making requests to the target.

Further, if the measurement nodes are sufficiently numerous, then sampling measurements would also test the load-management abilities of the target (if the latter has to service several thousand sampling requests every second).

We therefore use sampling measurements in our system, and note the round-trip time: the interval between sending a request and receiving a reply. Our samples have three parameters: *sampling type* is the kind of request made to the target, for instance a statically or dynamically generated resource might be requested; *sampling frequency* is how frequently we make requests; and *sampling interval* (or timeout) is the interval of time we wait for a reply. If a reply is not received during this interval then we give up and consider the system unresponsive within that interval.

The outputs of the measurement programs are appended to logs on the physical host that runs the virtual host on which the measurement program runs. These logs are gathered at the end of the experiment.

The timing in these measurements is assumed to be loose and approximate: we do not synchronise the measurement programs in any way to ensure that they all produce a request at exactly the same instant.

This also means that each time we run an experiment, the results may vary: but the conclusion from observations made during the experiment (i.e., whether the attack succeeds or not) does *not* change. It would not be sensible to engineer the system for perfect replication; the overheads would be unbearable since there are many sources of non-determinism and jitter in a typical commodity cluster. Instead we look for a compromise: the *qualitative* conclusions are exactly reproducible, but the quantitative measurements may vary.

In the example (§3) we used httping to request a (static) index page at a one-second frequency, and that times-out after one second (to avoid having overlapping samples).

4.2.3 Attack

An attack is started at some point during the experiment, after the measurement scripts have been started, and ended before measurement is stopped. We give attacks 3 attributes: *type* (which attack to run), *beginning* (when to start the attack), and *duration* (for how long to run the attack).

Attacks can be combined in sequence or in parallel, as the researcher chooses. We can start multiple (same or different) simultaneous attacks, or sequence them, as better suits the experiment. Thus step 2c in Fig. 1 can consist of multiple concurrent and sequential substeps, one for each attack.

Attacks are run on virtual hosts. The attacking hosts do not carry out any measurement. Currently a virtual host can only run a single attack at a time; thus simultaneous attacks must be started from different virtual hosts.

There may also be attack-specific parameters such as "intensity" but this could also be achieved by starting multiple simultaneous attacks of the same type.

Finally, we prefer to run attacking scripts rather than replay attack traffic, since this spares us from having to rewrite TCP traffic or modulate the traffic to better suit our link capacity (e.g., if the traffic was captured on a higher-speed link than the one being used in our setup).

In the example (§3) we ran a single attack, whose type was Slowloris, and which began 10 seconds into the experiment and lasted for 20 seconds.

4.2.4 Load polling

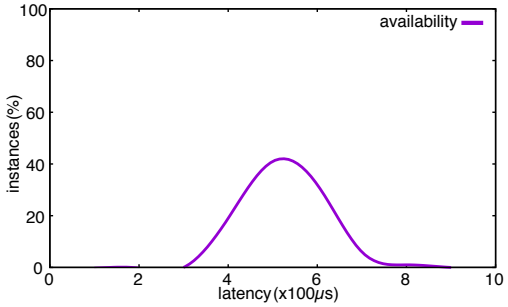
After measurement (§4.2.2), the second form of data acquisition consists of polling the load on hosts that carry out measurement, to assess whether they might be overloaded. This corresponds to step 3 in Fig. 1. We poll load since if measurement hosts are overloaded then their readings might be less credible.

We set a polling frequency (e.g., once every 5 seconds) to actively probe the physical host for CPU, memory and network saturation by examining related files in `/proc`. If load, memory usage, or network throughput appear abnormal (e.g., load is too high) then we lower the number of virtual hosts residing on that physical host, or the intensity of their activities.

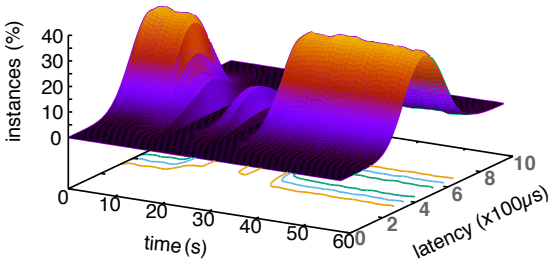
4.2.5 Finishing

At the end of the experiment, the measurement, proxy and target software are stopped – the attack would have been stopped earlier. The Docker instances are stopped too, and can be deleted.

All the logs are gathered from the physical machines (step 4 in Fig. 1), and downloaded into a machine that is used for analysis and graphing, described next.



(a) Latency histogram at time = 5*s*. Since this is a histogram, we expect that the curve will collapse during an attack (since we do not receive latency information from the overwhelmed target).



(b) We glue together latency histrograms (e.g., Fig. 4a) across time to produce this 3D graph, which gives us a visual summary of the experiment. You can see the distribution drop and fluctuate during the attack.

Figure 4: Graphical outputs from DoSarray

4.3 Output

We are now at step 5 in Fig. 1: analysing and graphing the data gathered during the experiment. The data consists of the outputs produced by the measurement scripts (§4.2.2) which has been appended to a file. The format and content of these scripts' output tends to vary between measurement programs.

We have one file for every virtual host carrying out measurements during the experiment. We run an analysis script that digests these files to produce a single file that contains values extracted from the original output.

The script that processes output from the measurement scripts must be adapted for each source of measurement (since they are likely to produce different output), but the graphing scripts can be reused entirely.

We produce two kinds of graphs: (i) *availability* and (ii) *round-trip latency*. Availability graphs are straightforward; an example was given in Fig. 2.

4.3.1 Latency graphs

Round-trip latency consists of measuring the time taken between issuing a request to the target and receiving the response. The measurement scripts are issuing such requests regularly.

This information is graphed as a histogram, an example of which is given in Fig. 4a for time = 5*s*, i.e., before the attack began.

We also build a histogram *over time*, as shown in Fig. 4b to produce a summary visualisation of the effect of the attack on the target, as witnessed by the measurement nodes.

Since accurately interpreting Fig. 4b can be challenging because of its dependence on the observer's perspective, we found it useful to project the graph into a contour map to more clearly identify variations in latency. An example of this is shown in Fig. 5.

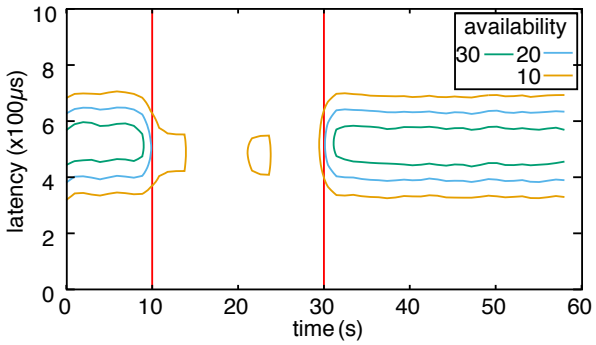


Figure 5: Contour plot of Fig 4b, showing the attack window.

In a glance the contour diagram tells us whether the attack succeeded *fully* (which produces a gap in the attack window, indicative of unavailability), or *partly* (evident by smaller blobs during the attack window, indicating reduced availability), and the variation in *average latency* (by seeing where the histogram peaks).

Thus the contour diagram became our main summary visualisation for experiments, and we use it to compare the effects of different attacks on different systems and mitigations in §6.

5 Evaluation

We deployed DoSarray in two locations: in a physical testbed and in a virtualised network of VMs (within a single server) during development. In this section we only report measurements obtained on a physical testbed, because of their better fidelity.

5.1 Scalability

We measure the effect on physical system load when we increase the number of virtual hosts. Ideally we would like to run large-scale experiments using the least quantity of physical resources necessary, and to this end it is useful to know whether there is a limit on how many virtual nodes can can be afforded for a particular kind of experiment (without compromising the quality of the results), and how the quality of results could be compromised if the limit is exceeded.

Our physical setup consists of 8 servers interconnected via 10GbE links to a switch, and each having an 8-core Intel Xeon E5-2630L 1.80GHz CPU and 64GB RAM on a Dell 072T6D v.A01 motherboard. We use Ubuntu Linux 14.04.5 LTS running v4.4.0-31 of the kernel.

For this experiment we repeat our example from §3 several times, each time using a network (§4.1) that includes an increasing number of virtual hosts V in each physical host, and vary $V \in \{15, 30, 60, 120, 250\}$ in each run of the experiment. We used a maximum V of 250 since we assigned a /24 virtual network to each physical host, but in principle one could assign larger subnets. Our physical network consists of 8 machines, therefore the total size of the experiment network is $1\,\text{physical} + V \times 7\,\text{virtual}$. Thus the total size of the network ranges from 106 (when $V = 15$) to 1751 (when $V = 250$).

We look for resource *saturation* or any indication that V might be too large for the available physical resources. We did see an increase in utilisation of all resources as we increased V from 15 to 250, but for all values of V we tried, there were no network errors, and none of the main resources (CPU, memory, and network capacity) were exhausted. The results are shown in Fig. 6 for $V = 250$. We repeat each experiment 3 times, and the graph shows the maximum, average, and minimum values across the three runs.

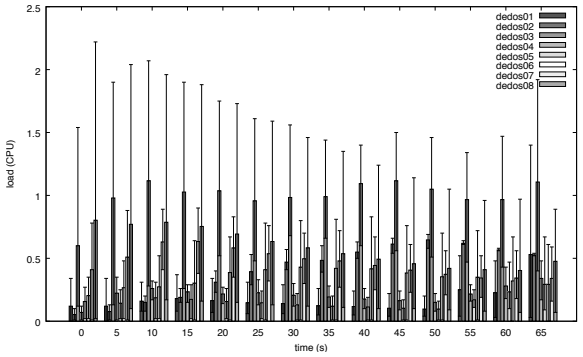
This suggests that the attack and measurement programs we used are not sufficiently active to exhaust our resources for $15 \leq V \leq 250$. Using multiple instances of the attacker, a larger V, and more resource-intensive measurement scripts would incur a greater load of course; polling load during experiments helps indicate whether the results obtained are trustworthy. We show an example of excess load in the next section.

5.2 Attack diversity

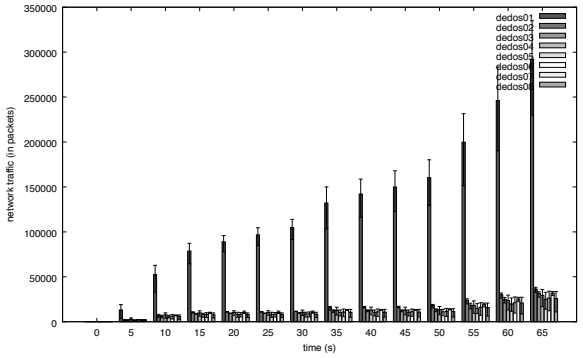
In DoSarray we can easily customise how attacks are carried out by modifying Bash scripts. This allows us to easily mix attacks – in sequence or in parallel – and run simultaneous diverse attacks on the target.

As an example of this, we rerun the 3 experiments for $V = 250$ from the previous section, but modified to perform 16 simultaneous attacks (rather than a single attack) and also increased the intensity of the attack by changing script-specific parameters (we set Slowloris to start $\times 10$ more connections and use shorter timeouts).

The resulting load is graphed in Fig. 7. Compared with Fig. 6a, this shows a much higher load, and it shows a saturation of our 8 CPU cores.



(a) CPU load



(b) Network counter value for packets received

Figure 6: CPU and network (Rx) load measured when overlaying physical hosts with 250 virtual hosts. The machine 'dedos01' is running the target (Apache with Event MPM); 'dedos02' runs the attack and 249 measurement instances, and the remaining machines all run 250 measurement instances.

5.3 Use and extensibility

DoSarray follows the Unix philosophy in consisting of a collection of tools, each geared for a specific purpose (for example, starting the Docker instances, graphing, etc).

We found it useful to automate the experiments to eliminate tedious and repetitive commands. Thus each run of a batch of experiments is started by calling a single script that sets up and carries out the experiments, downloads and processes the results, and some time later we are presented with the graphs.

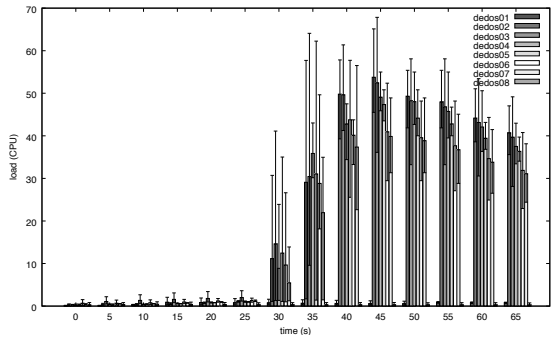


Figure 7: Graph showing high contention for CPU time.

At the end of each experiment, all the raw and processed data from that experiment, including resulting graphs, is kept inside a new directory for later reference. This helped us organise the data from over 100 past experiments we ran using DoSarray using a variation of parameters, and data from rerun experiments to test new features of DoSarray.

Changing and extending DoSarray consists of editing Bash scripts for the operations part, and Python for the analysis and graphing.

5.4 Target diversity

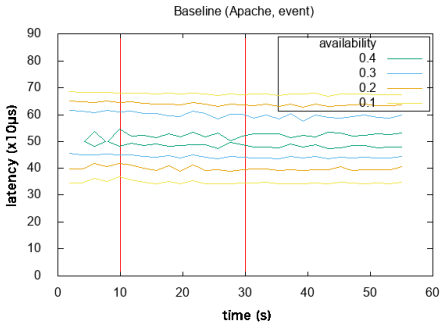
In this paper we only report on using DoSarray for experiments targetting HTTP, but we have also prepared container images for experiments targetting SIP. We have successfully used these images to run experiments, albeit manually. Setting up the experiments for SIP is more complex than for HTTP, since it involves pairing UAC and UAS nodes between which the calls will be started, rather than having the HTTP service be the focus of the experiment. But all of this adaptation is expressible in Bash or Python – for example, we also need to adapt the log parser to generate graphs – and nothing so far indicates that DoSarray cannot be extended to automate SIP-related experiments.

6 Survey of HTTP DoS

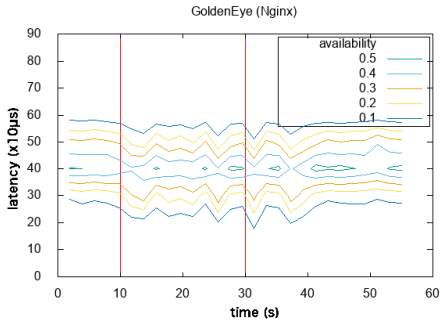
We used DoSarray to evaluate various attacks related to HTTP. In our experiments we used different attacks (Slowloris, GoldenEye and Tor's Hammer) to target HTTP servers (Apache, Nginx, lighttpd), proxies (Varnish and HAproxy), and mitigations (e.g., modreqtimeout, ModSecurity, iptables tuning).

In this section we describe a selection of our results, based on the contour diagramming method from §4.3.1.

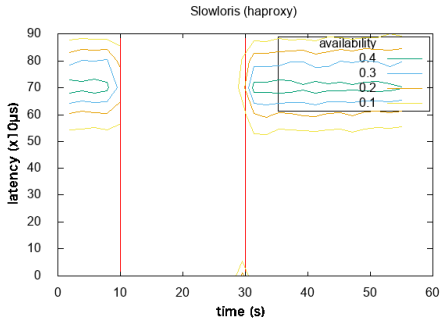
We begin by showing the baseline result for Apache, i.e., when it is not under attack. We observe that Apache (v2.4.26) continues to function smoothly during the window when an attack would have occurred:



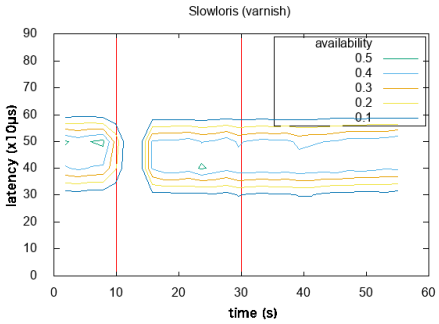
Next we look at Nginx v1.13.8 vs GoldenEye: we notice that Nginx is unaffected by the attack, and that the average latency is lower than that for baseline Apache:



Next we look at HAProxy v1.8.3 vs Slowloris, where we see both that it is affected by the attack, and that it has a higher response latency compared to both Nginx and Apache:



Finally we look at Varnish v5.2.1 vs Slowloris, where we see it is briefly affected by the attack, but quickly recovers.



7 Conclusion

The design and implementation of DoSarray appear to be sufficiently simple and general to accommodate DoS experiments on a large class of software. We opted to write most of DoSarray in Bash since we did not require sophisticated programming abstractions. DoSarray mostly involves doing job control, but specialised for the needs of running DoS experiments.

As we came to make DoSarray available to others we had the surprising realisation that we felt conflicted about making *everything* easily available: while we are content making the code available, we are reluctant to make the container images available since they contain attack scripts. These scripts were freely downloaded from the Internet, but we feel awkward about redistributing potentially harmful code further. We still have not arrived at a satisfactory solution that minimises bureaucracy and maximise sharing of such research artefacts.

We hope that DoSarray could be developed into an executable compendium of DoS experiments, spanning the full taxonomy described by Mirkovic et al. [16]. Directions for future research include adding targets and attacks for different application-layer protocols.

8 Acknowledgments

We thank Daniel Thomas, Joachim Breitner and the anonymous reviewers for feedback, and Zihao Jin and Pardis Pashakhanloo for their help with configuring attack scripts. This work is supported in part by the the Defense Advanced Research Projects (DARPA) under Contract No. HR0011-16-C-0056, and NSF grants CNS-1513679 and CCF-1763514.

9 Availability

The DoSarray scripts, documentation, and the data for the example used in this paper, are available at `https://www.seas.upenn.edu/~nsultana/dosarray/`

References

[1] Booters, Stressers and DDoSers. `https://www.incapsula.com/ddos/booters-stressers-ddosers.html`. Accessed: 2018-05-09.

[2] DDoS-for-Hire Service Webstresser Dismantled. `https://krebsonsecurity.com/2018/04/ddos-for-hire-service-webstresser-dismantled/`. Accessed: 2018-05-09.

[3] AHRENHOLZ, J. Comparison of CORE Network Emulation Platforms. *IEEE MILCOM* (2010), 166–171.

[4] BAKER, M. 1,500 scientists lift the lid on reproducibility. *Nature 533*, 7604 (May 2016), 452–454.

[5] BEGLEY, C. G. Reproducibility: Six red flags for suspect work. *Nature 497* (May 2013), 433–434.

[6] CHADHA, R., BOWEN, T., CHIANG, C., GOTTLIEB, Y., POYLISHER, A., SAPELLO, A., SERBAN, C., SUGRIM, S., WALTHER, G., MARVEL, L., NEWCOMB, A., AND SANTOS, J. CyberVAN: A Cyber Security Virtual Assured Network Testbed. *IEEE MILCOM* (2016), 1125–1130.

[7] CLARK, B., DESHANE, T., DOW, E. M., EVANCHIK, S., FINLAYSON, M., HERNE, J., AND MATTHEWS, J. N. Xen and the Art of Repeated Research. In *USENIX Annual Technical Conference, FREENIX Track* (2004), pp. 135–144.

[8] HANDIGOL, N., HELLER, B., JEYAKUMAR, V., LANTZ, B., AND MCKEOWN, N. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (2012), ACM, pp. 253–264.

[9] KRISHNAMURTHI, S., AND VITEK, J. The Real Software Crisis: Repeatability As a Core Value. *Commun. ACM 58*, 3 (Feb. 2015), 34–36.

[10] MANCO, F., LUPU, C., SCHMIDT, F., MENDES, J., KUENZER, S., SATI, S., YASUKATA, K., RAICIU, C., AND HUICI, F. My VM is Lighter (and Safer) Than Your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 218–233.

[11] MIRKOVIC, J., BENZEL, T. V., FABER, T., BRADEN, R., WROCLAWSKI, J. T., , AND SCHWAB, S. The DETER Project: Advancing the Science of Cyber Security Experimentation and Test. *IEEE International Conference on Technologies for Homeland Security* (Nov. 2010).

[12] MIRKOVIC, J., E.ARIKAN, WEI, S., FAHMY, S., THOMAS, R., AND REIHER, P. Benchmarks for DDoS Defense Evaluation. In *Proceedings of the IEEE AFCEA MILCOM* (2006).

[13] MIRKOVIC, J., FAHMY, S., REIHER, P., AND THOMAS, R. K. How to Test DoS Defenses. In *Conference for homeland security, 2009. CATCH'09. Cybersecurity applications & technology* (2009), IEEE, pp. 103–117.

[14] MIRKOVIC, J., FAHMY, S., REIHER, P. L., AND THOMAS, R. K. Automating DDoS Experimentation. In *Proceedings of the USENIX DETER workshop* (2007).

[15] MIRKOVIC, J., HUSSAIN, A., FAHMY, S., REIHER, P., AND THOMAS, R. K. Accurately Measuring Denial of Service in Simulation and Testbed Experiments. *IEEE Transactions on Dependable and Secure Computing 6*, 2 (2009), 81–95.

[16] MIRKOVIC, J., AND REIHER, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review 34*, 2 (2004), 39–53.

[17] MIRKOVIC, J., REIHER, P., FAHMY, S., THOMAS, R., HUSSAIN, A., SCHWAB, S., AND KO, C. Measuring Denial of Service. In *Proceedings of the 2nd ACM workshop on Quality of protection* (2006), ACM, pp. 53–58.