The Journal of Open Source Software

Magicwand: A platform to provide high-quality, reliable, and reproducible data sets for low-and-slow DDoS attacks.

Banjo Obayomi[1], **Christopher H. Todd**[1], **Lucas Cadalzo**[1], **David Killian**[1], **and Anthony C. Wong**[2]

1 Two Six Labs **2** Unaffiliated

DOI: 10.21105/joss.03032

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @nidhog
- @gradvohl

Submitted: 30 November 2020
Published: 20 May 2021

Summary

`Magicwand` is a Python platform to provide high-quality, reliable, and replicable data sets for low-and-slow distributed denial-of-service (LSDDoS) attacks. Using a Command line interface (CLI), researchers can generate packet capture (.pcap) data with labels for attackers and benign clients. With the use of Docker, and JavaScript Object Notation (JSON) files, `Magicwand` provides a customizable and extensible data generation framework to generate a multitude of network traffic PCAPs.

Statement of Need

Distributed denial-of-service (DDoS) attacks remain a pervasive cybersecurity threat (Krebs, 2016), (Plante, 2015), and (Markoff, 2008). While various commercial services such as Akamai and CloudFlare are effective in mitigating traditional, high-volume DDoS attacks, LSDDoS attacks present a different kind of threat (Makrushin, 2013). These attacks stealthily degrade server performance through cleverly crafted transmissions of data, rather than brute-force floods of traffic. The adversary sends low-bandwidth requests that slowly transmit data and consume a disproportionate amount of the server's resources. The relatively small amount of traffic needed to cause a denial-of-service has two important consequences: these attacks are both easier to launch and more difficult to detect, in comparison to high-volume DDoS.

In order to create a robust LSDDoS defense, researchers in the security community must have access to sufficient data capturing how these attacks behave. The most widely-adopted public source of data for LSDDoS was generated by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick (Sharafaldin & Ghorbani, 2018). However, the vast majority of instances in this dataset are from high-volume attacks, and the dataset does not contain any benign traffic directed towards the victim host. Additionally, the range of LSDDoS attacks employed is limited. In an effort to facilitate the generation of more thorough LSDDoS datasets, we are introducing our traffic generation tool `Magicwand`.

Through its development, these key goals informed the design and implementation:

- The user must not be required to be an expert in low-and-slow DDoS attacks.
- Data generated must be representative of low-and-slow DDoS attack behavior and their impact on victim servers.
- Data generated should be validated through a series of automated tests and checks.
- The software must be able to generate data on any hardware that meets a minimum set of requirements.

- The software should be extensible to enable new components in the future.
- The software should allow for user flexibility in configuration of LSDDoS scenarios.

Extensibility

Magicwand was designed as a framework that can be extended based on project needs; to this end, the tool is comprised of extensible *components* that can be considered a foundation for expanding the tool's capabilities. The four core components of the system are the attack clients, benign clients, systems under threat (SUTs), and sensors. Each of these components was designed to be operationalized into any Magicwand experiment.

Attack Clients

The set of implemented attack images consists of Slowloris (Valialkin, 2014), R U Dead Yet (Shekyan, 2011), Slow Read (Shekyan, 2011), Apache Killer (Stampar, 2011), Sockstress (Hornby, 2012), and a second implementation of Slowloris (Valialkin, 2014). The JSON files provide configurations that can scale the intensity of attack and adjust other attack-specific-specific parameters (e.g. threads per connection for Apache Killer).

Benign Clients

For benign traffic generation, Magicwand uses customized agents, based on Locust (Byström et al., 2019), that continuously browse the SUT, traversing randomly from link to link within the SUT. The JSON files allow for the configuration of parameters such as clients per container, connection runtime, rate of activity, and more.

System Under Threat (SUT)

The primary SUT included with Magicwand is an Apache WordPress server. We choose this stack for two reasons: first, it is widely used on the web, ensuring our system has wide applicability. Second, there is a substantial set of known LSDDoS attack implementations for this stack.

Sensors

Magicwand includes sensors that use tcpdump to capture real-time network traffic and provide a PCAP for each experiment. An additional sensor captures Round-Trip-Time (RTT) data of the SUT to understand the impact of the attack and what a client would experience during the experiment.

Developing A New Component

Developing a new Magicwand component requires extending the Python MwComponent abstract base class. The critical methods/attributes are validate, run, and config and must be implemented for a new component. The developer must also create a configuration.json file that defines the parameters for the component and default values.

Component Containerization

Magicwand experiments rely on Docker containers to run any component. For each component, a user will need to create a Dockerfile that will run the component during the experiment, a docker-compose.yml file that will handle parameters and runtime-config for the component during the experiment.

Experiments

Experiments are defined by the components that will run, the timeline for each component to execute, and customizable configuration for each component. Each experiment will run locally and generate data that includes PCAPs of the network traffic generated, data from each sensor component, and metadata from the experiment.

System Calibration

To ensure the data is representative of LSDDoS attacks and realistic benign clients, Magicwand includes calibration functionality.

For benign components, Magicwand ensures that the amount of benign traffic flowing towards the SUT is not so large that it causes congestion, absent any malicious traffic.

For attack components, Magicwand finds the configuration required to ensure an attack is strong enough to induce a denial-of-service condition on the SUT. For example, Apache Killer calibration process will determine the parameters needed to yield a mean RTT greater than 2x the mean RTT of experiments with only benign traffic.

Attack calibration works by launching two experiments: one with only benign traffic and the other with only attack traffic. We then compare various metrics between the two experiments that detail the state of the SUT, such as RTT and memory consumption. The SUT parameters are held constant between each experiment to compare how the attack and benign traffic changes based on the inputs. If the ratio between the metrics of the benign and attack experiments indicates that the attack succeeded in inducing a denial-of-service condition, then the JSON configuration file is saved.

Data Validation

We validate each experiment to ensure representative and expected data was produced. After each experiment, Magicwand conducts a series of checks based on the input parameters against the generated data and run logs. Each component has its own validation functionality.

Attack components verify that each attack yielded expected signatures in the data; an example would be Apache Killer generating packets that send overlapping bytes in the resulting PCAP. Each attack has specific validations that are tailored to how the attack works.

Benign client validation confirms that the clients performed the tasks they were configured to do; an example is verifying that an experiment with 10 benign clients attempted to spin up 10 connections during the experiment.

Sensor validation confirms that the sensors ran and yielded the data each sensor is responsible for and that the data is not corrupted. SUT validation confirms that SUT receives all expected trafic and that it ran as expected through the experiment.

While we have been systematic and methodical in crafting these checks to make them as comprehensive as possible, these are not exhaustive. This is where researchers can define validation methods to ensure their `Magicwand` experiments are producing data that they would expect.

Acknowledgements

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) under Contract # HR0011-16-C-0060. This document was cleared for release under Distribution Statement" A" (Approved for Public Release, Distribution Unlimited). The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense of the U.S. Government.

References

Byström, C., Heyman, J., Hamrén, J., & Heyman, H. (2019). Locust. In *GitHub repository*. https://github.com/locustio/locust; GitHub.

Hornby, T. (2012). Sockstress. In *GitHub repository*. https://github.com/defuse/sockstress; GitHub.

Krebs, B. (2016). *KrebsOnSecurity hit with record DDoS*. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

Makrushin, D. (2013). *The cost of launching a DDoS attack*. https://securelist.com/the-cost-of-launching-a-ddos-attack/77784/

Markoff, J. (2008). Before the gunfire, cyberattacks. *The New York Times*. https://www.nytimes.com/2008/08/13/technology/13cyber.html

Plante, C. (2015). *Valve's $18 million dota 2 tournament delayed by DDoS attack*. https://www.theverge.com/2015/8/4/9097597/the-international-dota-2-ddos-attack-valve

Sharafaldin, L., I., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *SciTePress*. https://doi.org/10.5220/0006639801080116

Shekyan, S. (2011). SlowHTTPTest. In *GitHub repository*. https://github.com/shekyan/slowhttptest; GitHub.

Stampar, M. (2011). KillApachePy. In *GitHub repository*. https://github.com/tkisason/KillApachePy/; GitHub.

Valialkin, A. (2014). Goloris. In *GitHub repository*. https://github.com/valyala/goloris; GitHub.