Creating a Large-scale Memory Error IoT Botnet Using NS3DockerEmulator

Islam Obaidat*, Bennett Kahn†, Fatemeh Tavakoli*, and Meera Sridhar*
*Department of Software and Information Systems, UNC Charlotte, Charlotte, NC, USA
{iobaidat, ftavakol, msridhar}@uncc.edu
†Department of Mathematics and Computer Science, Tulane University, LA, USA
bkahn1@tulane.edu

***Abstract*—DDoSim, a simulation testbed for mimicking real-world, large-scale botnet DDoS attacks, is presented. DDoSim offers various capabilities, including running user-specified software, testing botnet-recruitment exploits, and measuring the severity of resulting DDoS attacks. DDoSim leverages `NS3DockerEmulator`'s Docker and NS-3 integration to load Docker containers with actual binaries and connect them over a simulated NS-3 network. DDoSim is validated through a comparison with results from real hardware experiments.**

This paper focuses on the results of an experiment series concerning deploying a memory error botnet on IoT devices. Unlike the Mirai attack, which relies on default credentials, these experiments exploit memory error vulnerabilities to access IoT devices. DDoSim also implements realistic IoT churn, reflecting dynamic network conditions in real-world IoT environments. The results reveal that memory error vulnerabilities enable botnet recruitment, while network conditions, attack size, and duration all have a proportional impact on target servers. DDoSim is publicly available for researchers' use.

***Index Terms*—DDoS Simulation, Botnet, Large-scale, IoT**

I. Introduction

Botnets, networks of compromised devices controlled by an attacker, pose significant threats as they frequently execute *Distributed Denial-of-Service* (DDoS) [1] attacks, disrupting or incapacitating target servers or networks with illegitimate traffic. DDoS attacks account for approximately two billion dollars in annual financial damages [2]. High-profile botnet incidents, such as Mirai [3], [4], Mozi [5], and Kraken [6] highlight the need for a thorough understanding of these attacks. To fully understand botnet-based attacks, it is essential for researchers to simulate them using real-world binaries and analyze their various aspects, including entry-point vulnerabilities, the proportion of infiltrated devices, and attack severity. While prior studies have often focused on mathematical models or specific attack phases, to our knowledge, no existing framework offers realistic modeling and analysis capabilities for botnet DDoS attacks, particularly in response to the emergence and evolution of new threats.

In this paper, we present DDoSim (*Distributed Denial of Service Simulator*), a framework for simulating and assessing large-scale botnet DDoS attacks. DDoSim enables researchers to create simulated environments comprising potential bot devices running user-specified binaries, a malicious attack node equipped with malware of the user's choice, and a simulated server acting as the DDoS target. This framework accurately and effectively models all phases of botnet DDoS attacks while maintaining scalability and cost-efficiency. Moreover, DDoSim permits real-time analysis and investigation of botnet DDoS attacks at any stage, allowing users to quantify attack severity (e.g., a server's *average received data rate*—the overall data received at a target, averaged over numerous time steps), assess botnet magnitude (e.g., the proportion of compromised devices), and scrutinize compromised devices (e.g., examine the backdoor vulnerability). Researchers can also utilize DDoSim to implement and evaluate defense strategies against these attacks in the simulated environment, measuring their effectiveness in mitigating or preventing exploits.

We build DDoSim upon `NS3DockerEmulator` [7], a network emulator that integrates Docker [8] (a container-based virtualization technology) and the NS-3 network simulator [9] to emulate large-scale networks. `NS3DockerEmulator` is designed to create a simulated wireless network consisting of hundreds of ad hoc nodes (nodes that communicate directly without needing a central unit to organize that communication). In `NS3DockerEmulator`'s setup, the ad hoc nodes utilize Docker to run a simple program that broadcasts a "HelloWorld" message through the simulated wireless network, intended to generate substantial data for analysis.

Unlike the original setup and design purpose of `NS3DockerEmulator`, DDoSim leverages the integration of NS-3 and Docker to replicate large-scale botnet DDoS attacks on a simulated network with actual binaries loaded in Docker containers. DDoSim allows users to seamlessly deploy a number of nodes that represent specific components in a botnet DDoS attack. Specifically, DDoSim has three components: Attacker, Devs, and TServer. Attacker is a Docker node representing an attacker, loaded with tools and scripts to remotely exploit and control a large number of devices for attacking a target server. Devs are a variable number of Docker nodes loaded with actual binaries that represent the devices targeted by the attack. TServer is a customized NS-3 node that represents the target of a botnet DDoS attack.

We conduct an experiment series using DDoSim to show its capabilities and address pivotal research questions related to its functionalities: (R1) Are *memory error vulnerabilities* a feasible attack vector for infiltrating *Internet-of-Things* (IoT)

This research was supported in part by NSF DGE award #1947295

devices and facilitating botnet recruitment?; (R2) What proportion of targeted IoT devices can an attacker successfully recruit to become a constituent bot within a botnet by exploiting such vulnerabilities?; (R3) How do factors such as the number of devices flooding a target, attack duration, and dynamic IoT network conditions affect the magnitude of a botnet DDoS attack? Regarding (R1), prior botnets have capitalized on simpler vulnerabilities, such as dictionary attacks, to compromise a device using default credentials. However, with recent legislative measures mandating vendors to equip devices with reasonable security levels [10]–[12], it is conceivable that attackers will utilize more sophisticated vulnerabilities, such as memory error-related vulnerabilities.

In our experiment series, we utilize widely common binaries in IoT devices as Devs. We represent Devs as IoT devices since IoT devices are the main targets of recent botnet attacks—the reasons being that IoT devices are ubiquitous, connected to the Internet, often share similar vulnerabilities, and frequently lack basic security protections [13], [14]. Consequently, in our experiment series, we consider Devs to represent IoT devices specifically and select a data rate in our simulated network that aligns with the average data rate of IoT devices.

With this focus on IoT devices, we deploy Devs running `Connman` [15] and `Dnsmasq` [16] (two network management daemons prevalent in real-world IoT devices), which contain known memory error vulnerabilities [17], [18] (reported as CVE-2017-12865 and CVE-2017-14493, respectively) that can be remotely exploited. Furthermore, we create attack scripts (hosted in Attacker) to gain backdoor access to Devs by exploiting these vulnerabilities. Then, these scripts install the open-source, readily-available Mirai malware [19]–[21] on the compromised Devs, forming a botnet. We utilize this botnet to launch a DDoS attack against TServer and subsequently measure the magnitude of the ensuing damage on TServer. Lastly, in order to more accurately model a real-life IoT network, we implement network *churn* (i.e., when devices unpredictably leave and rejoin the network) [22] in some of our experiments and compare results with and without churn.

Through our experiment series, we draw the following answers to our research questions: (R1 Answer) memory error vulnerabilities *do* serve as a feasible attack vector for infiltrating IoT devices and facilitating botnet recruitment; (R2 Answer) an attack leveraging memory error vulnerabilities as a backdoor in IoT devices successfully recruits all targeted devices (a 100% infection rate), effectively making them constituent bots within a botnet; and (R3 Answer) the magnitude of a botnet DDoS attack is influenced by factors such as the number of devices flooding a target and the attack duration, which both increase the severity of the attack, while dynamic IoT network conditions tend to reduce the attack's severity.

To validate DDoSim, we perform experiments in a real-life setting, using a physical computer acting as Attacker to infect Raspberry Pis loaded with `Connman` or `Dnsmasq` (serving as Devs) and directing them to perform a botnet DDoS attack against a second physical computer acting as TServer. These components communicate over a network created using an actual router, in which Devs are connected wirelessly to this network, and the other two physical computers are connected using Ethernet connections. We obtain similar results when conducting experiments in this real-life setting as we do by performing experiments in DDoSim with the same parameters, thus demonstrating the validity of DDoSim.

The contributions of our work include the following:

- We develop DDoSim, a framework that utilizes `NS3DockerEmulator`'s `Docker` and `NS-3` integration to perform cost-effective, scalable, and customizable botnet DDoS attack simulations with user-chosen, real-world binaries over a simulated network. DDoSim supports an in-depth analysis of all stages of the attack.
- We use DDoSim to design and deploy an IoT botnet attack experiment series involving `Connman`, `Dnsmasq`, and Mirai malware. We report on the results of this attack on a target server; we implement network churn that mimics the dynamic conditions of realistic IoT networks.
- We validate DDoSim by conducting experiments on actual hardware configured identically to some of our DDoSim experiments, demonstrating DDoSim validity through consistent and comparable results.
- We publish our framework online at https://github.com/sridhar-research-lab/DDoSim for the research community to perform experiments, develop, and test defenses to mitigate future botnet DDoS attacks.

The rest of the paper is organized as follows: Section II overviews DDoSim. Section III reports on our experiment series using DDoSim. Section IV details our experimental setup, results, useful insights, and DDoSim's validation. Section V demonstrates DDoSim's use cases and discusses the challenges and limitations of our work. Section VI outlines related work, and Section VII concludes our work.

II. The DDoSim Framework

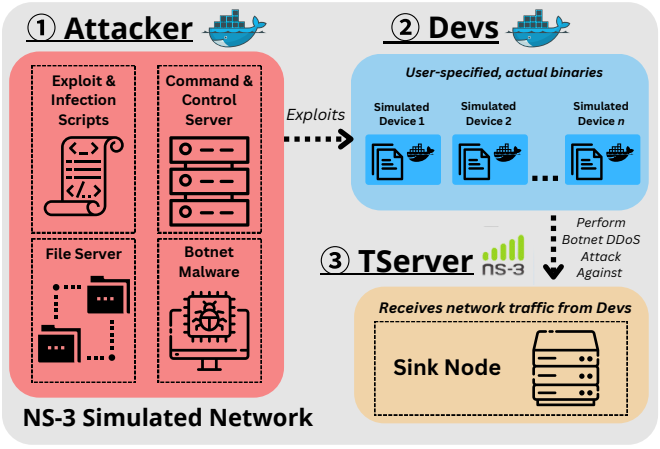

Fig. 1. DDoSim Overview

In this section, we present an overview of our DDoS simulation framework, DDoSim. DDoSim leverages the integration of NS-3 and Docker provided by `NS3DockerEmulator` to construct its main components, which are in line with a typical

botnet DDoS attack scenario; the components are *Attacker*, *Devs*, and *TServer*, as shown in Figure 1. Attacker (Component ①) in DDoSim is a Docker container node that hosts binaries and tools to recruit Devs as bots. Devs (Component ②) represent a collection of network-facing devices (such as *Internet-of-Things* (IoT) devices, smartphones, and computers) that we create in DDoSim using actual binaries hosted in Docker containers. TServer (Component ③) is an NS-3 node representing the target server on which Attacker executes a botnet DDoS attack using the compromised Devs.

In DDoSim, Attacker, Devs, and TServer communicate through a simulated NS-3 network, which we can customize to be any network (e.g., wired LAN). To accurately reflect contemporary and future networks, many of which are transitioning from the legacy IPv4 to IPv6 [23], we incorporate IPv6 communication support in DDoSim, even though `NS3DockerEmulator` does not inherently provide it.

The following subsections discuss DDoSim's components.

A. Attacker

Attacker aims to exploit vulnerabilities in Devs and use them to create a bot army to attack TServer. Typically, an attacker in real-world scenarios has multiple elements: a person (or an entity) and servers (or services) with dedicated software tools utilized to conduct the attack. In DDoSim, we consider all of these elements to be a single component, *Attacker*, which we represent as a Docker node. Attacker consists of four sub-components: Exploit & Infection Scripts, Botnet Malware, Command and Control Server, and File Server.

Exploit & Infection Scripts: These scripts exploit vulnerabilities in Devs to download and execute architecture-specific Botnet Malware on them, facilitating Devs' use as bots in DDoS attacks. These Exploit & Infection Scripts are tailored to the specific vulnerability that Attacker targets in Devs. Additionally, these scripts can modify passwords and activate `telnet`/`ssh` (if disabled) in the compromised Devs.

Botnet Malware: Botnet Malware connects the compromised Devs to C&C Server, rendering their primary function to await commands from the server. Furthermore, Botnet Malware can simultaneously scan the network for new potential victims and listen for commands from C&C Server [13].

Command and Control Server: In line with a real-life botnet DDoS attack [24], Devs infected with Botnet Malware report to C&C Server. This server records the available bots and can issue commands to these bots, instructing them to perform a botnet DDoS attack against TServer.

File Server: Attacker requires a File Server to host Exploit & Infection Scripts and Botnet Malware necessary to mount a botnet DDoS attack.

Network Configuration: In DDoSim, we use a Docker node type to implement Attacker. This node consists of two main parts: a Docker container and an NS-3 ghost node. The Docker container hosts and runs the components discussed above. The NS-3 ghost node, in turn, serves as a proxy for Attacker to send and receive data over the simulated network. In this manner, the Docker container must be able to communicate with the NS-3 node. To do so, the Docker container transmits data through one of its Ethernet network devices, `eth0`. A Linux virtual Ethernet device (`veth`) acts as a bridge between the container's `eth0` and the NS-3 ghost node's `TapBridge NetDevice`, a special type of `NetDevice` in NS-3 that receives data from another application via a bridge. The NS-3 ghost node, with the help of the `veth` bridge, gives an illusion to the Docker container that it is connected directly to the NS-3 simulated network.

B. Devs

Devs simulate a variety of network-facing devices, such as desktop computers, servers, and IoT devices, with users having the option to select the number of these simulated devices. While emulating entire systems using QEMU [25] and connecting them to the NS-3 network via virtual bridges is possible [26], doing so on a large scale requires significant processing powers, which limits DDoSim's scalability. To circumvent this limitation, Devs use lightweight Docker containers loaded with the user-selected software (e.g., a network-facing program) that is part of the network-facing devices, effectively mimicking the crucial aspects of network-facing devices necessary for botnet recruitment. Additionally, DDoSim accommodates diverse binary architectures (e.g., MIPS, ARM) for Devs using Docker Buildx [27].

In general, representing IoT devices as Docker containers does not affect the accuracy of DDoSim (see §V-C), as a device's susceptibility to botnet recruitment is predominantly determined by the vulnerability of its network-facing program. As such, running the other components of network-facing devices is both unnecessary and resource-intensive. Given these considerations, the user-selected software in Devs should fulfill two main requirements: (a) having one or more exploitable vulnerabilities and (b) processing input from the network, allowing Attacker to deliver exploits to Devs remotely.

Network Configuration: Similar to Attacker, we use Docker nodes to represent Devs. Devs' Network Configuration is identical to that of Attacker, except that each simulated device has its own Docker container and NS-3 ghost node combination.

C. TServer

TServer represents the target server against which Devs attempt to execute a botnet DDoS attack when instructed by Attacker. Specifically, Devs flood TServer with illegitimate traffic upon receiving the attack command from C&C Server.

Network Configuration: In DDoSim, we use an NS-3 node to represent TServer, where we implement a customized sink application capable of receiving data transmitted from any source within the simulated network. The NS-3 simulator provides a `Node` class that connects to its simulated network, allowing for the transmission, reception, and processing of data within the simulation. This `Node` class, along with its other associated classes (e.g., the `Application` class), can be customized, facilitating comprehensive analysis across various network layers, thereby providing users the flexibility to examine packets and a wide assortment of network metrics.

III. DDoSim Experiments Series With A Mirai Memory Error IoT Botnet

In this section, we report on the customization of DDoSim's components for investigating DDoS attacks using memory error botnet recruitment in IoT devices, as well as our setup for the NS-3 simulated network.

A. Attacker

In this subsection, we discuss the customization of Attacker's subcomponents to exploit `Connman` and `Dnsmasq` (IoT binaries used in Devs, detailed in the next subsection) *stack-based buffer overflow* vulnerabilities (CVE-2017-12865 and CVE-2017-14493, respectively), recruiting Devs as bots, ultimately orchestrating a botnet DDoS attack against TServer.

Exploit & Infection Scripts: These scripts exploit `Connman` and `Dnsmasq`'s vulnerabilities to gain backdoor access into Devs. English et al. [28] present *Return Oriented Programming* (ROP; an exploitation technique that chains together short sequences of existing code) exploit against `Connman`'s vulnerability that acts as a malicious DNS server to deliver a payload that spawns a shell in a target device. However, we desire one payload that downloads and installs Botnet Malware on a target device. Thus, we create a shell script that performs this task and host this script on File Server at the URL `ShellScript_URL`. We then modify English et al.'s payload, changing it from spawning a shell to making the system call `execlp("sh\0","sh\0","-c","curl -s ShellScript_URL | sh",NULL)`. This system call both downloads and runs our shell script on the target device.

For Devs running `Dnsmasq`, we use an attack strategy nearly identical to the one discussed above. However, `Dnsmasq`'s vulnerability lies in the functionality responsible for handling DHCPv6 RELAYFORW messages. Consequently, we craft a RELAYFORW DHCPv6 message that contains the above payload and send it to Devs.

Botnet Malware: As discussed earlier, our Exploit & Infection Scripts deliver ROP payloads to Devs to trigger, download, and execute Botnet Malware. In our experiment series, we use Mirai malware to control the compromised Devs because Mirai's source code is open source, and its effectiveness has been proven [13]. After infecting the victim device, Mirai malware hides its presence by obfuscating its process name and removing the downloaded malware binary. Also, this malware attempts to kill processes associated with other DDoS variants and processes bound to port 22 or 23 (TCP) to fortify itself [13].

Command and Control Server: Once Botnet Malware executes on the compromised Devs, it connects Devs to Attacker's C&C Server. In our experiment series, Attacker's Docker node uses C&C Server provided with Mirai's published code [19], [20]. Once we start our simulator, we can access C&C Server from a terminal via telnet to monitor the connected bots and instruct them to attack TServer.

File Server: In our experiment series, we install an Apache server on Attacker's Docker node to host our malicious binaries and scripts to deliver them to Devs upon request.

B. Devs

We load each Devs' Docker container with either `Connman` [28] or `Dnsmasq` [16], two lightweight network management daemons, binaries because they are prevalent in the IoT realm [29]–[32] and are network-facing programs; therefore, as mentioned in §I, we treat Devs in our experiment series to represent *only* IoT devices. Another reason for choosing `Connman` and `Dnsmasq` is the presence of known memory error vulnerabilities [28], [33] (stack-based buffer overflow) in them. While these vulnerabilities have been patched in newer versions, thousands of IoT devices run outdated software tools, including these two programs [28], [34]–[39].

To mimic a variety of IoT devices, we set up the Docker containers of Devs with different memory protection levels. Specifically, each binary in Devs enables some subset of `W⊕X` [40], [41] (Write XOR Execute; memory regions are either writable or executable, but not both) [38], [39] and ASLR (Address Space Layout Randomization; randomizing memory addresses of a program) [42]. Consequently, our attack model assumes that Attacker cannot perform code injection [28] or return-to-libc attacks [28] due to these defenses in Devs. Furthermore, we assume that Attacker can access Devs' binaries and analyze them to construct working ROP payloads. We also assume that Devs use the vulnerable versions of our selected network management daemons. Lastly, we assume that a large number of Devs (IoT devices in real-world scenarios) have the same binary structure (which is a reasonable assumption since a significant number of binaries are reused across products and vendors [37]).

Our experiments *mimic* actual IoT devices by using Docker containers loaded with software of interest from IoT firmware for scalability reasons. However, with more powerful hardware, DDoSim can perform complete emulation of IoT firmware using Firmadyne [43] (which leverages QEMU for full IoT firmware emulation) and connect it to the NS-3 network using virtual bridges [44], as mentioned in §II-B. Additionally, while DDoSim supports different architectures for Devs (using Docker Buildx), for our experiments, we load Devs' Docker containers with either `Connman` or `Dnsmasq` binaries exclusively for the x86 (64-bit) architecture.

C. TServer

In our experiment series, all bots attack TServer as soon as they receive the attack command from C&C Server. Here, we attack TServer with Mirai's volumetric UDP-PLAIN flood attacks, a botnet DDoS attack supported by Mirai to flood a target with UDP packets. For this purpose, we customize our implementation of TServer to receive data packets from the compromised Devs and then log the overall size of the received data packets in each simulation run, i.e., TServer records the magnitude of each attack simulation for further analysis.

D. Simulated Network

NS-3 simulated networks can be customized to represent different network types (such as LAN or WAN with wired and/or wireless connectivity). In our experiment series, we

aim to create a simulated network that represents the Internet and then connect DDoSim's components to this simulated Internet. In a typical botnet DDoS attack scenario carried out over the Internet, the connection between two components in DDoSim (e.g., Attacker and Devs) consists of different hubs (e.g., home routers and ISP switches) connected together using different mediums (e.g., fiber optics and WiFi connections). Conceptually, we can represent this Internet connection link as a single connection line with specific latency and bandwidth. Therefore, we create a simulated NS-3 network that connects each of DDoSim's components together over an Ethernet connection link. Because we aim to simulate IoT devices only, we choose a 100-500 kbps data rate, as this is an average range for such devices in real life based on several studies [45]–[47].

IV. IoT Botnet Evaluation and Validation

In this section, we present the experimental setup for our experiments series outlined in §III. We then discuss their results and provide insights derived from these experiments. Lastly, we demonstrate DDoSim's validation experiments.

A. IoT Botnet: Experimental Setup

In our reported experiment series, we use a laptop with 16 GB memory and a 2.7 GHz Intel Core i5 CPU. We run DDoSim on a virtual machine with Ubuntu 22.04 LTS as the guest OS. We implement DDoSim with Docker version 20.10 and NS-3 simulator version 3.37. Due to the hardware limitations, we conduct experiments with up to 200 Devs.

Establishing a DDoS attack against TServer in DDoSim requires an initialization phase to prepare each component to perform its required task successfully. DDoSim begins by creating and building Docker containers for Attacker and Devs. After building and starting these containers, DDoSim connects them to the virtual network interfaces and bridges in order to connect them to the NS-3 simulator. Next, DDoSim starts the NS-3 simulation to create the network and TServer and then connects Attacker and Devs to this network. Here, the NS-3 simulator requires the user to specify a time duration for each simulation run, during which it creates and runs all network simulation components, including network topologies and packets. After this time period, NS-3 terminates the network simulation and reclaims the resources used. For all experiments presented in DDoSim, we set the NS-3 simulation time to 600 seconds.

After DDoSim is initialized, the simulated network begins operation, and each node carries out its designated tasks. Following is a detailed discussion of the execution flow for each component within the framework.

Attacker: Once Attacker is connected to the NS-3 network, it starts its subcomponents that require initialization: C&C Server, Apache server, and Exploit & Infection Scripts. We can access C&C Server remotely via `telnet` to issue commands to control the bots. The Apache server serves the `ShellScript` and Mirai malware binaries upon request from the compromised Devs. On the other hand, DDoSim starts a malicious DNS server to listen for DNS requests from Devs to respond with malicious DNS responses to exploit `Connman`'s vulnerability. Similarly, a DHCP Python script runs and periodically sends malformed DHCPv6 messages to exploit `Dnsmasq`'s vulnerability. Here, we send the DHCPv6 messages to a multicast IPv6 address since the vulnerability in `Dnsmasq` resides in its IPv6 processing module, and there is no broadcast address in IPv6 [48].

Devs: Devs running `Connman` send DNS requests to the Attacker's malicious DNS server, while those running `Dnsmasq` accept DHCPv6 messages from any source. Thus, all Devs receive our malicious payloads, which turn them into bots. Using C&C Server, we direct all bots to perform a volumetric UDP-PLAIN flood attack against TServer.

TServer: TServer receives all attack traffic from bots and records the attack's magnitude for further analysis.

To more accurately mimic a realistic IoT network, our experiments incorporate churn, the unpredictable departure and return of Devs in a network. Various algorithms exist in the literature for simulating churn. In our work, we adopt Fan et al.'s approach [22], designed specifically for IoT networks, where high churn rates in Devs result from their unreliable connections and insufficient energy. Fan et al.'s algorithm defines a device's *leaving factor* ($L(h)$) as a function of its link quality and energy supply, with $L(h) = (1 - q(h))(1 - e(h))$, where h is a host, $q(h) \in [0, 1]$ represents link quality, and $e(h) \in [0, 1]$ denotes remaining energy. In our experiments, we randomly assign q and e values to each device in Devs. Next, the algorithm assigns a *leaving probability* to each device as:

$$l(h) = \begin{cases} \varphi_1 L(h), & \text{if } L(h) \leq 0.4 \\ \varphi_2 L(h), & \text{if } 0.4 < L(h) \leq 0.7 \\ \varphi_3 L(h), & \text{if } L(h) > 0.7 \end{cases} \quad (1)$$

where $\varphi_1, \varphi_2, \varphi_3 \in [0, 1]$ are coefficients reflecting variable leaving behaviors of a device. For the experiments in [22], the authors use 0.16, 0.08, and 0.04 for φ_1, φ_2, and φ_3, respectively. We use these same values in our experiments.

In our work, we implement two variations of the above-mentioned algorithm: *static churn* and *dynamic churn*. In static churn, devices leave the network with probability p = $l(h)$ at the simulation's outset and do not rejoin. Conversely, dynamic churn re-estimates p for each device every 20 seconds, enabling intermittent departures and rejoining. This approach more accurately mimics real-world scenarios where devices rejoin the network upon condition improvement (e.g., reconnecting to a power source after energy depletion).

B. IoT Botnet: Preliminary Experimental Results

We run our experiments (discussed in §IV-A) using DDoSim several times, varying either the number of devices in Devs or the attack duration across each run. For each run, we measure the TServer's *average received data rate*, which is the received data rate at TServer during one second, averaged over the entirety of the attack duration. The received data rate for second i, in turn, is the total size of the packets (in kbps) received by TServer from Devs during second i. Thus, average received data rate, $D_{received}$, is defined as follows:

$$D_{received} = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} d_{j,i}}{n} \quad (2)$$

where n is the attack duration (in seconds), m is the number of devices in Devs, and $d_{j,i}$ is the amount of traffic received by TServer from device j (i.e., Devs = $\{ device\_j \mid j \leq m \}$) during second i (in kilobits).

Average received data rate of varying the number of Devs: In our initial experiments, we investigate the impact of varying the number of Devs (10-150) and churn levels (no churn, static churn, and dynamic churn) on the average received data rate at TServer. In these experiments, once Devs connect to C&C Server, we initiate a 100-second attack on TServer. Figure 2 illustrates the results of these experiments, representing the average received data rate at TServer.

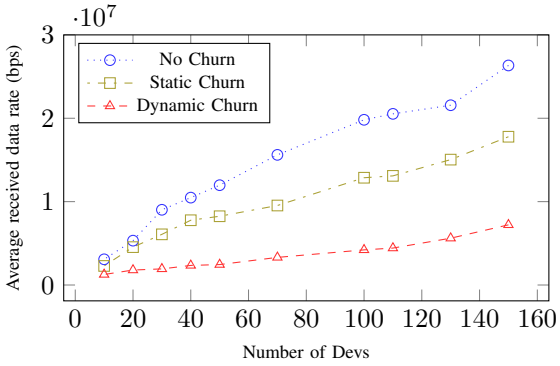

Fig. 2. Results of varying number of Devs in DDoSim experiments

The results in Figure 2 show a non-linear increase in the results as Devs increase, regardless of churn level[1], which can be attributed to congestion and collisions stemming from elevated network traffic [49], [50]. The highest average received data rate occurs in the no churn experiments, where all Devs persist throughout the simulation. Static churn experiments exhibit a reduced average received data rate due to Devs' departure. Dynamic churn experiments depict the lowest average received data rate, as Devs have multiple opportunities to exit the network (even during the attack phase), and those disconnected before issuing the attack command would not participate even upon rejoining (due to missing the attack command).

Average received data rate of varying the attack duration: To see how the duration of an attack affects the average received data rate, we perform four rounds of attacks. In each round, we conduct three attacks, varying the attack duration (150, 200, and 300 seconds). We perform attack rounds that vary the number of Devs (50, 100, 150, 200) to ensure our trend is independent of the number of Devs. Figure 3 shows that increasing the attack duration consistently increases the average received data rate, as captured by TServer.

Memory and time costs during each run: We conduct other experiments with varying numbers of Devs (20, 40, 70, 100, and 130) to investigate the resources utilized by DDoSim during each run, maintaining a 100-second attack duration. In these experiments, we record memory usage and actual attack time associated with different simulation phases.

[1]Churn is exclusively used in the experiments reported in this figure. All other experiments utilize the no churn approach.

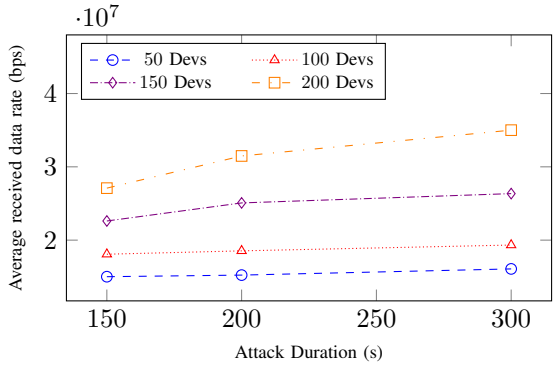

Fig. 3. Results of varying attack duration in DDoSim experiments

We measure memory usage in two simulation phases: Pre-attack Mem and Attack Mem. We measure Pre-attack Mem after initializing all Docker containers and starting NS-3 when network traffic minimally impacts memory due to limited packet exchanges between Devs and Attacker. On the other hand, we measure Attack Mem during the attack phase when Devs flood TServer, where DDoSim requires more memory due to the large packet volume generated in the simulation.

As we see in Table I, Pre-attack Mem increases with a higher number of Devs, consuming memory for storing binaries running on Devs. Furthermore, Attack Mem surpasses Pre-attack Mem due to the large number of packets during the DDoS attack. For instance, 130 Devs require 1.79GB (3.11GB - 1.32GB) extra memory to store traffic generated during the attack. These results demonstrate the impact of the number of Devs on the memory usage during various phases of the simulation, particularly when a DDoS attack is in progress.

TABLE I
HARDWARE RESOURCES CONSUMED BY DDOSIM

Devs	Pre-attack Mem (GB)	Attack Mem (GB)	Attack Time (m:ss)
20	0.38	0.39	2:03
40	0.52	1.15	2:43
70	0.73	1.47	3:22
100	0.94	1.93	3:48
130	1.32	3.11	5:14

We also measure the actual attack duration (Attack Time in Table I), which captures that actual time it takes from the beginning of the attack until it ends. As seen in Table I, Attack Time increases with a higher number of Devs, as more devices generate additional traffic, resulting in increased processing and task handling by the simulator. Due to limited hardware resources in our experiments, task queuing is necessary, resulting in extended Attack Times. Consequently, Attack Time exceeds the specified 100-second simulated attack duration.

C. IoT Botnet: Useful Insights

Apart from allowing us to measure metrics as the above, DDoSim can provide researchers with information and insights about DDoS attacks that can aid in understanding and analyzing these attacks to build and create defenses in the target system. Following is a brief discussion of some of the useful insights we deduce from our conducted experiments.

- In our experiment series, we observe that Attacker leveraged the `curl` command to download Mirai malware in the compromised Devs. This information supplies us with hints that can be used to build/test better defenses against these dangerous attacks. For instance, firmware vendors may choose not to allow or install the `curl` command or similar commands in Linux-based IoT devices.
- In our experiment series, we notice the impact of the devices' data rate, which directly affects the magnitude of the DDoS attacks. In some IoT devices (e.g., smart temperature sensors), it is recommended to limit the available data rate on these devices since a limited data rate is sufficient to operate such devices and would decrease the magnitude of the DDoS attacks if such an approach is adopted at scale.
- In our DDoS attack experiments (and similarly in other real-world DDoS attacks), attackers leverage an entry point, which is common/similar in a large number of IoT devices. For instance, the Mirai attack leveraged similar default credentials to access and compromise IoT devices. Reducing the similarities in IoT devices, such as default credentials and different binary structures, prevents attacks from compromising IoT devices at scale.

D. IoT Botnet: Framework Validation

To validate DDoSim, we conduct the same experiment series in two scenarios: one on actual physical hardware and another using DDoSim. If DDoSim generates results comparable to those from the actual hardware scenario, we consider DDoSim validated, demonstrating that it performs as intended and closely simulates the behavior of actual hardware.

In the first scenario, we utilize multiple Raspberry Pi 3 Model B devices, two desktop computers, and a Netgear Nighthawk X6 router. We configure Raspberry Pis (which run Raspbian OS) as Devs and randomly load them with vulnerable `Connman` or `Dnsmasq` binaries. Hence, these Raspberry Pis run `Connman` or `Dnsmasq` as their network management daemon. We set up both desktops with Ubuntu 22.04 LTS, with one functioning as TServer and the other as Attacker. We install Wireshark on TServer to analyze the traffic, while Attacker's desktop hosts its sub-components, including C&C Server. Using the Netgear router, we create a local network for communication between components (TServer, Attacker, and Devs). We wirelessly connect Devs (with data rates limited to 100-500kbps to mimic the actual bandwidth of IoT devices) and establish Ethernet connections for the desktops.

In this setup, Devs send requests to Attacker, which replies with malicious payloads, exploiting vulnerabilities and infecting Devs with Mirai malware. Once all Devs connect to Attacker, we issue Mirai's UDP-PLAIN flood attack against TServer, while Wireshark captures incoming traffic for further analysis, allowing us to observe the attack's impact on TServer.

In the second scenario, we use DDoSim to replicate the same experiment with identical settings as in the first scenario, which uses actual hardware. We conduct multiple experiments for both scenarios, varying the number of Devs (1-19). During these runs, we calculate the average received data rate at TServer, allowing for a comparative analysis of the results.

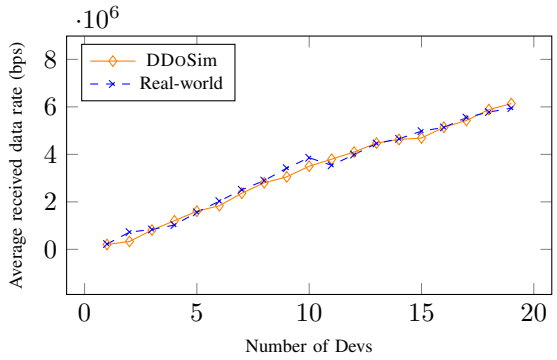

Fig. 4. Results of real-world vs. DDoSim experiments

Figure 4 shows the results obtained from both experiment series discussed above. This figure illustrates that the average received data rate for both experiment series is similar, indicating that DDoSim accurately replicates experimental outcomes comparable to those performed on actual hardware. This finding supports the validation of DDoSim as an effective tool for simulating botnet DDoS attacks.

V. Discussion

A. DDoS*im Use Cases*

We envision DDoSim to be used for different purposes by the research community, including the following two possible use cases: 1) testing/validating proposed defense strategies and 2) testing mathematical models of botnet spread.

1) Testing or Validating Defense Strategies: Numerous studies propose techniques for detecting and defending against DDoS attacks. As demonstrated in [51], a significant number of these works suggest *machine learning* (ML) based defenses, leveraging supervised and unsupervised learning algorithms. Most ML-based DDoS detection or mitigation approaches rely on extracting features from incoming network traffic (e.g., IP address, traffic rate) and feeding them into an ML model, such as neural networks, to classify the traffic as malicious or benign. Additionally, several proposed methods for DDoS attack defense or detection are blockchain-based [52], [53], which can be categorized into network-level mitigation and near-attack domain location.

Use with DDoSim: DDoSim enables the extraction of network traffic at any layer. One example use case is testing a defense strategy by generating both malicious DDoS and normal traffic to TServer, followed by analyzing incoming traffic using an ML model to assess the classification model's

accuracy. Another use case involves generating large traffic datasets or enriching existing ones with DDoSim to train ML models for DDoS traffic detection.

2) Testing Mathematical Models of Botnet Spread: Many studies propose models or predictions related to the propagation of botnet malware. Generally, these works use epidemic modeling techniques, such as the *Susceptible-Infected-Recovered* model [54]–[61]. Epidemic models are typically a system of *ordinary differential equations* (ODEs); as such, researchers typically provide *simulations* and predictions of botnet spread using typical methods for solving ODE systems.

Use with DDoSim: While mathematical models offer general insights into system behavior, actual behavior often differs from these predictions. DDoSim provides a cost-effective framework for simulations that more closely resemble real-life botnet propagation. Researchers can run experiments in DDoSim and extract the number of infected devices in Devs at any time step, enabling them to assess whether these more realistic simulations align with their models.

B. Challenges

We face multiple challenges during the adaptation of `NS3DockerEmulator` to develop DDoSim. First, running `Dnsmasq` (and exploiting its vulnerability) requires the use of IPv6, which is not supported in `NS3DockerEmulator`; therefore, we modify DDoSim at multiple locations to add IPv6 support so that we can deliver our ROP exploit to `Dnsmasq`'s IPv6 module through our simulated network. Second, in attempting to use `NS3DockerEmulator` cleaning routines (code dedicated to reclaiming used resources) in DDoSim, we encounter several issues, such as the entire framework crashing before we can instantiate our network. We implement several changes to the cleaning routines' code to avoid these bugs. Third, `NS3DockerEmulator` only allows Docker container nodes; for TServer, however, we must be able to analyze several statistics related to its networking accurately. While we could install a package like WireShark on a potential TServer Docker node, such tools do not provide easy analysis of application-level statistics. We overcome this challenge by implementing TServer as an NS-3 node.

C. Limitations

In our conducted experiments, we mimic IoT devices using user-chosen binaries in Docker containers. It is possible that real devices running these binaries may have various unforeseen security measures that render an exploit unsuccessful on that device. Moreover, in DDoSim, all components share uniform connections, while real-world factors like distance and network quality impact device-device links. Although we partially address dynamic IoT network conditions using churn, actual conditions may diverge from our churn implementation. Furthermore, in many ways, DDoSim simplifies the attack phase. For example, in our exploit against `Connman`, we manually configure Devs to listen to our malicious DNS server. In the real world, an attacker would need to trick devices into listening to their server (e.g., using a malicious link [62]).

VI. Related Work

Previous studies focus on simulating the Mirai attack against actual IoT devices and exploring metrics, such as infection and spread rates. For instance, Kelly et al. [63] present a simulated network to replicate the Mirai attacks on IoT devices, demonstrating the ease of infection. Tanaka et al. [64] simulate Mirai attacks in various network topologies to illustrate the malware's infection rate. Adan [65] simulates the Mirai infection process on a Raspberry Pi. Yamaguchi et al. [66], [67] mathematically simulate the infection rate of Mirai and similar botnets using agent-oriented Petri net models (PN^2).

Our work is distinct from these works in that we develop a fully customizable botnet simulation framework that requires no physical devices and allows researchers to examine and analyze any framework component at any time. With our framework, researchers can efficiently run multiple experiments to investigate the impact of specific features on botnet attacks. Moreover, in our experiments discussed in §III, we demonstrate that memory error vulnerabilities can provide backdoor access to devices, while other discussed works use default credentials to gain backdoor to target devices.

Other works have simulated general DDoS attacks; for instance, Karthik and Shah [68] simulate a DDoS attack against a cloud-hosted server, demonstrating a significant increase in server computation time during the attack. Furfaro et al. [69] present a simulation model for DDoS attacks, using it to test the effectiveness of various attack features. Our work stands out as our simulated DDoS attack is part of a larger, customizable framework that allows for modifications in other components (e.g., binaries in Devs). These features enable researchers to investigate how adjusting these components affects the actual DDoS attack.

VII. Conclusion

In this paper, we leverage `NS3DockerEmulator` to develop DDoSim, a framework for simulating and analyzing botnet DDoS attacks. Using DDoSim, we conduct an experiment series to mimic IoT botnet DDoS attacks by remotely exploiting memory error vulnerabilities in simulated IoT devices (running real-world IoT binaries), granting us backdoor access to these devices. Once we have access, we infect the compromised devices with Mirai malware binaries. We then instruct the compromised devices to launch a DDoS attack on a target server. Moreover, we implement static and dynamic churn in DDoSim to closely mimic real-world IoT networks. Based on these experiment series, our findings reveal that (1) attack magnitude increases with device count and attack duration, (2) network churn reduces attack severity, and (3) memory error vulnerabilities provide viable entry points for botnet recruitment. Additionally, we conduct two other experiments, one using DDoSim and the other on actual hardware, with the similarity in results highlighting the validity of our framework. Lastly, we publish the source code of DDoSim for the research community on GitHub: https://github.com/sridhar-research-lab/DDoSim.

REFERENCES

[1] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[2] Neustar, "Worldwide DDoS attacks and cyber insights research report," Neustar, 2017. [Online]. Available: https://www.discover.neustar/201705-Security-Solutions-DDoS-SOC-Report-LP.html

[3] L. H. Newman, "What we know about Friday's massive East Coast Internet outage," Wired, 2016. [Online]. Available: https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/

[4] L. E. S. Jaramillo, "Malware detection and mitigation techniques: lessons learned from Mirai DDoS attack," *Journal of Information Systems Engineering & Management*, vol. 3, no. 3, p. 19, 2018.

[5] R. Lakshmanan, "Mozi IoT botnet now also targets Netgear, Huawei, and ZTE network gateways," TheHackerNews, 2021. [Online]. Available: https://thehackernews.com/2021/08/mozi-iot-botnet-now-also-targets.html

[6] S. Simon, "Meet Kraken: A new Golang botnet in development," Zerofox, 2022. [Online]. Available: https://www.zerofox.com/blog/meet-kraken-a-new-golang-botnet-in-development/

[7] J. A. A. Aldana, "NS3DockerEmulator," GitHub, 2017. [Online]. Available: https://github.com/chepeftw/NS3DockerEmulator/

[8] Docker, Inc., "Docker: Empowering app development for developers," Docker, 2022. [Online]. Available: https://www.docker.com/

[9] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[10] P. Stockburger, "Decoding "reasonableness" under California's IoT law," Docker, 2021. [Online]. Available: https://www.dentons.com/en/insights/articles/2021/april/7/decoding-reasonableness-under-californias-iot-law

[11] US Congress, "H.R.1668 - Internet of Things cybersecurity improvement act of 2020," Congress.gov, 2020. [Online]. Available: https://www.congress.gov/bill/116th-congress/house-bill/1668/text

[12] European Commission, "The EU cybersecurity act," European Commission, 2021. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/cybersecurity-act

[13] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai botnet," in *Proceedings of the 26th USENIX Security Symposium (USENIX)*, 2017, pp. 1093–1110.

[14] Q. He, C. Wang, G. Cui, B. Li, R. Zhou, Q. Zhou, Y. Xiang, H. Jin, and Y. Yang, "A game-theoretical approach for mitigating edge DDoS attack," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[15] M. Holtmann, "ConnMan," The Linux Kernel Archives, 2008. [Online]. Available: https://git.kernel.org/pub/scm/network/connman/connman.git/

[16] S. Kelley, "Dnsmasq," TheKelleys, 2001. [Online]. Available: https://thekelleys.org.uk/dnsmasq/doc.html

[17] CVEDetails, "CVE-2017-12865 detail," National Vulnerability Database, 2017. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-12865/

[18] CVEDetails, "CVE-2017-14493 detail," National Vulnerability Database, 2017. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-14493/

[19] B. Krebs, "Source code for IoT botnet 'Mirai' released," Krebs on Security, 2016. [Online]. Available: https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/

[20] Anna-Senpai, "Mirai botnet," GitHub, 2016. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[21] Jsong302, "Mirai botnet: Modified version," GitHub, 2018. [Online]. Available: https://github.com/jsong302/mirai

[22] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Churn-resilient task scheduling in a tiered IoT infrastructure," *China Communications*, vol. 16, no. 8, pp. 162–175, 2019.

[23] L. Ubiedo, T. O'Hara, M. J. Erquiaga, and S. Garcia, "Current state of IPv6 security in IoT," Stratosphere Research Laboratory, 2020. [Online]. Available: https://www.stratosphereips.org/blog/2020/11/04/white-paper-current-state-of-ipv6-security-in-iot

[24] A. Marzano, D. Alexander, O. Fonseca, E. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. C. Chaves, Ítalo Cunha, D. Guedes, and W. Meira, "The evolution of Bashlite and Mirai IoT botnets," in *Proceedings of the 23rd IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 813–818.

[25] F. Bellard, "QEMU, a fast and portable dynamic translator." in *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[26] N. Duan, N. Yee, B. Salazar, J.-Y. Joo, E. Stewart, and E. Cortez, "Cybersecurity analysis of distribution grid operation with distributed energy resources via co-simulation," in *Proceedings of the IEEE Power & Energy Society General Meeting (PESGM)*, 2020, pp. 1–5.

[27] Docker, "Buildx," GitHub, 2020. [Online]. Available: https://github.com/docker/buildx/

[28] K. V. English, I. Obaidat, and M. Sridhar, "Exploiting memory corruption vulnerabilities in Connman for IoT devices," in *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 247–255.

[29] J. Edge, "Jolla: Sailfish OS, Qt, and open source," LWN, 2013. [Online]. Available: https://lwn.net/Articles/561463/

[30] V. Savov, "Samsung drops Android for Tizen in new Gear 2 smartwatches," TheVerge, 2014. [Online]. Available: https://www.theverge.com/2014/2/22/5437150/samsung-drops-android-for-tizen-in-new-gear-2-smartwatches

[31] A. D. Venecia, "Sailfish OS: Everything you have to know," CellularNews, 2020. [Online]. Available: https://cellularnews.com/mobile-operating-systems/sailfish-os-everything-you-have-to-know/

[32] Shodan, "Shodan: Search engine for the Internet of Everything," Shodan, 2021. [Online]. Available: https://www.shodan.io/search?query=dnsmasq

[33] Google Security Research, "Security research PoCs," GitHub, 2017. [Online]. Available: https://github.com/google/security-research-pocs/tree/master/vulnerabilities/dnsmasq

[34] A. Cui, M. Costello, and S. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, 2013, pp. 1–13.

[35] F. Xiao, L.-T. Sha, Z.-P. Yuan, and R.-C. Wang, "VulHunter: A discovery for unknown bugs based on analysis for known patches in industry Internet of Things," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 267–279, 2017.

[36] A. Mohanty, I. Obaidat, F. Yilmaz, and M. Sridhar, "Control-hijacking vulnerabilities in IoT firmware: A brief survey," in *Proceedings of the 1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec)*, 2018.

[37] R. Yu, F. Del Nin, Y. Zhang, S. Huang, P. Kaliyar, S. Zakto, M. Conti, G. Portokalidis, and J. Xu, "Building embedded systems like it's 1996," in *Proceedings of the 29th Annual Network and Distributed System Security Symposium (NDSS)*, 2022, pp. 1–18.

[38] I. Nadir, H. Mahmood, and G. Asadullah, "A taxonomy of IoT firmware security and principal firmware analysis techniques," *International Journal of Critical Infrastructure Protection*, vol. 38, p. 100552, 2022.

[39] B. Zhao, S. Ji, J. Xu, Y. Tian, Q. Wei, Q. Wang, C. Lyu, X. Zhang, C. Lin, J. Wu, and R. Beyah, "A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2022, pp. 442–454.

[40] S. Andersen and V. Abella, "Changes to functionality in Microsoft Windows XP service pack 2," Microsoft Corporation, 2004. [Online]. Available: https://download.microsoft.com/documents/australia/technet/winxpsp2.doc

[41] T. de Raadt, "OpenBSD 3.3," OpenBSD, 2003. [Online]. Available: https://www.openbsd.org/33.html

[42] PaX Team, "PaX ASLR (address space layout randomization)," The PaX Team, 2004. [Online]. Available: https://pax.grsecurity.net/docs/aslr.txt

[43] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for Linux-based embedded firmware." in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*, 2016, pp. 1–16.

[44] S. Späthe, B. Schirrmeister, and F. Geyer, "Simulation and benchmarking of IoT device usage scenarios using Zephyr and Qemu," in *Proceedings of the 9th International Conference on Smart Cities, Systems, Devices and Technologies (ICNS)*, 2020, pp. 6–11.

[45] S. S. I. Samuel, "A review of connectivity challenges in IoT-smart home," in *Proceedings of the 3rd MEC International conference on big data and smart city (ICBDSC)*, 2016, pp. 1–4.

[46] F. Muteba, K. Djouani, and T. Olwal, "A comparative survey study on LPWA IoT technologies: Design, considerations, challenges and solutions," *Procedia Computer Science*, vol. 155, pp. 636–641, 2019.

[47] F. Samie, L. Bauer, and J. Henkel, "IoT technologies for embedded computing: A survey," in *Proceedings of the 11th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, 2016, pp. 1–10.

[48] D. G. Chandra, M. Kathing, and D. P. Kumar, "A comparative study on IPv4 and IPv6," in *Proceedings of the 3rd International Conference on Communication Systems and Network Technologies (CSNT)*, 2013, pp. 286–289.

[49] R. Malhotra, V. Gupta, and D. R. Bansal, "Simulation & performance analysis of wired and wireless computer networks," *Global Journal of Computer Science and Technology*, vol. 11, no. 3, 2011.

[50] M. A. Alsmirat, Y. Jararweh, I. Obaidat, and B. B. Gupta, "Internet of surveillance: a cloud supported large-scale wireless surveillance system," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 973–992, 2017.

[51] A. Aljuhani, "Machine learning approaches for combating distributed denial of service attacks in modern networking environments," *IEEE Access*, vol. 9, pp. 42 236–42 264, 2021.

[52] R. Chaganti, B. Bhushan, and V. Ravi, "A survey on Blockchain solutions in DDoS attacks mitigation: Techniques, open challenges and future directions," *Computer Communications*, vol. 197, pp. 96–112, 2023.

[53] Z. A. Khan and A. S. Namin, "A survey of DDoS attack detection techniques for IoT systems using Blockchain technology," *Electronics*, vol. 11, no. 23, 2022.

[54] B. K. Mishra, A. K. Keshri, D. K. Mallick, and B. K. Mishra, "Mathematical model on distributed denial of service attack through Internet of Things in a network," *Nonlinear Engineering*, vol. 8, no. 1, pp. 486–495, 2019.

[55] M. T. Gardner, C. Beard, and D. Medhi, "Using SEIRS Epidemic Models for IoT botnets attacks," in *Proceedings of the 13th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2017, pp. 1–8.

[56] D. Acarali, M. Rajarajan, N. Komninos, and B. B. Zarpelão, "Modelling the spread of botnet malware in IoT-based wireless sensor networks," *Security and Communication Networks*, vol. 2019, pp. 1–14, 2019.

[57] H. Xia, L. Li, X. Cheng, X. Cheng, and T. Qiu, "Modeling and analysis botnet propagation in social Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7470–7481, 2020.

[58] D. Acarali, M. Rajarajan, and N. Komninos, "Modelling botnet propagation in networks with layered defences," in *Proceedings of the 5th International Symposium on Networks, Computers and Communications (ISNCC)*, 2018, pp. 1–6.

[59] G. Barakat, B. Al-Duwairi, M. Jarrah, and M. Jaradat, "Modeling and simulation of IoT botnet behaviors using DEVS," in *Proceedings of the 13th International Conference on Information and Communication Systems (ICICS)*, 2022, pp. 42–47.

[60] V. Q. Rufino, L. P. De Aguiar, D. S. Menasche, C. Lima, I. Cunha, E. Altman, R. El-Azouzi, F. De Pellegrini, A. Avritzer, and M. Grottke, "Beyond herd immunity against strategic attackers," *IEEE Access*, vol. 8, pp. 66 365–66 399, 2020.

[61] K. O. Chee, M. Ge, G. Bai, and D. D. Kim, "IoTSecSim: A framework for modelling and simulation of security in Internet of Things," *TechRxiv*, 2022, preprint.

[62] J. Kirk, "Advertising-based cyberattacks hit BBC, New York Times, MSN," ComputerWorld, 2016. [Online]. Available: https://www.computerworld.com/article/3044565/advertising-based-cyberattacks-hit-bbc-new-york-times-msn.html

[63] C. Kelly, N. Pitropakis, S. McKeown, and C. Lambrinoudakis, "Testing and hardening IoT devices against the Mirai botnet," in *Proceedings of the International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1–8.

[64] H. Tanaka and S. Yamaguchi, "On modeling and simulation of the behavior of IoT malwares Mirai and Hajime," in *Proceedings of the 21st IEEE International Symposium on Consumer Electronics (ISCE)*, 2017, pp. 56–60.

[65] M. F. O. Adán, "Designing an Internet of Things attack simulator," Bachelor's thesis, Metropolia University of Applied Sciences, 2019.

[66] S. Yamaguchi, "Botnet defense system: Concept and basic strategy," in *Proceedings of the 38th IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 1–5.

[67] S. Yamaguchi, H. Tanaka, and M. A. B. Ahmadon, "Modelling and evaluation of mitigation methods against IoT malware Mirai with agent-oriented Petri net PN2," *International Journal of Internet of Things and Cyber-Assurance*, vol. 1, no. 3-4, pp. 195–213, 2020.

[68] S. Karthik and J. Shah, "Analysis of simulation of DDoS attack in cloud," in *Proceedings of the International Conference on Information Communication and Embedded Systems (ICICES)*, 2016, pp. 1–5.

[69] A. Furfaro, G. Malena, L. Molina, and A. Parise, "A simulation model for the analysis of DDoS amplification attacks," in *Proceedings of the 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim)*, 2015, pp. 267–272.