Generation of Distributed Denial of Service Network Data with Phyton and Scapy

Stefan Görtz
Computer Science and Mathematics
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
stefan1.goertz@st.oth-regensburg.de

Sebastian Fischer
Computer Science and Mathematics
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
sebastian.fischer@oth-regensburg.de

Rudolf Hackenberg
Computer Science and Mathematics
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
rudolf.hackenberg@oth-regensburg.de

***Abstract*—Distributed Denial of Service attacks are among the most common and widespread network attacks. Due to their nature, they are difficult to defend. Intrusion detection systems, based on machine learning, are a promising approach to counter this threat. But to train these systems, data sets with Distributed Denial of Service attacks are needed. An implemented Python program, which creates Denial of Services packets and simulates distributed sending by multithreading, is presented. Unlike synthetically generated data with the use of simulators, real network traffic is generated. This eliminates errors and offers a better basis of data, as machine learning algorithms need data that is as error-free as possible in order to learn efficiently.**

***Keywords*—DDoS; Scapy; Synflood; Udpflood; PCAP; IDS.**

I. Introduction

Distributed Denial of Service (DDoS) attacks are among the most common network-based attacks and cause major economic damage. In the third quarter of 2022, the number of DDoS attacks increased sharply by 90%, compared to the third quarter of the previous year [1]. Internet of Things (IoT) devices are particularly vulnerable to these attacks, as they are accessible via the Internet and usually exchange data with cloud servers.

Detecting DDoS attacks is difficult, because the attack packets are indistinguishable from ordinary network traffic [2] and the packages come from many different sources. An intrusion detection system (IDS) is usually used to detect these attacks. However, conventional IDSs are no longer sufficient for more sophisticated attacks. The use of an artificial intelligence-based intrusion detection system (iIDS) is a promising approach to detect DDoS attacks. But for the development of an iIDS, attack data is needed to train the iIDS' machine learning modules. Since real data is not easily available and not classified, generated data is mostly used. Therefore, this paper presents an approach to generate real training data using Python and Scapy. It aims at a possible solution to the following question: How can DDoS network data on IoT devices suitable for machine learning be generated?

The paper is structured as follows: in section 2, the related work is introduced, in section 3 the theoretical foundations are shown, then our methods are presented in section 4 and the results are shown in section 5. Section 6 contains the discussion of the results. Subsequently, in section 7, we draw a conclusion and in the end in section 8, an outlook is given.

II. Related Work

Singh et al. [3] use the network simulation software NS2 to generate a network of 8 clients. They generate network traces of attack data, which consists injected packets, mixed with an attack based on a DDoS dataset. Through this dataset, a DDoS attack of 130 attack hosts on one target is generated for the duration of 50 seconds with a maximum throughput rate of 90,000 bps.

Arora and Dalal [4] use NET stress, a network stressing tool to introduce DDoS Attacks. They generate TCP, as well as UDP connections and carry out UDP flood and IGMP Flood attacks for the duration of 50 secs and 38 secs while benchmarking the network performance.

Baly et al. [5] use the graphical networksimulator GNS3 to create a topology of a webserver, three computers and one intruder with a Kali Linux distribution. The intruder carrys out DDoS attacks, captured by wireshark as pcap files.

Alzahrani and Hong [6] use the OMNET++ network simulator to generate DDoS attack network traffic in a simulated cloud environment. They generate various DDoS scenarios: Synflooding, HTTP flooding and UDP flooding. In addition, they generate non-intrusive traffic.

In contrast to the synthetically generated network traffic, with the use of simulators, we generate real network traffic. Generating network data synthetically has some potential sources of error. These are eliminated by performing the DDoS attacks in a real network environment. Furthermore, we can conduct attacks with longer durations and more individual attack hosts.

III. Theoretical Foundations

This section lays the theoretical introduction for DDoS data generation with Python using our method.

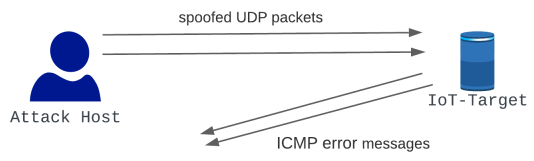

Fig. 1. UDP flooding

A. Denial of Service

Denial of Service (DoS) attacks aim to make a service unavailable, that is accessible via the Internet, unusable for legitimate visitors [7]. A distinction is made between two different attack categories, on the one hand attackers can aim to overload the network and hardware resources, for example bandwidth, memory and CPU cycles of the target, on the other hand, the behavior of the network protocols used, for example TCP or UDP, can be abused [8]. In the case of a successful attack, the entire resources of the attack target or protocol instance are consumed by the attacker and legitimate requests can no longer be processed [9].

B. Distributed Denial of Service

DDoS attacks use the DoS techniques with the help of many attack hosts. The attack hosts are devices, accessible through the Internet, which are infected with malicious code and involuntarily participate in the network attacks [8].

C. Anatomy of DDoS Attacks

Most DDoS attacks (64% in 2022) exploit the Transmission Control Protocol (TCP), followed by the User Datagram Protocol (UDP) with 22% [10]. The motivation of most DDoS attacks is political or economic. In so-called ransom DDoS attacks, cybercriminals blackmail their target into paying a ransom or holding out the prospect of a DDoS attack. The most commonly targeted industry segments include the aviation industry, government institutions, telecom facilities and media houses. The most frequent attacks occurred at intervals of ten to twenty minutes, followed by attacks lasting one to three hours [11].

D. UDP Flooding

UDP defines a minimal network protocol for connectionless data transfer with no guarantee of complete and correct data transfer. Applications create an UDP header for their data and transfer it via Internet Protocol (IP).

This DDoS attack variant uses a large number of packets to exhaust the hardware resources of the recipient. Accordingly, UDP flooding attacks belong to the first DDoS category. To carry out the attack, a high volume of UDP packets with spoofed IP address is sent to random ports of the target. UDP implementation on the receiver side searches for the corresponding application that accepts UDP packets on the addressed port. If no application is found, it responds with an Internet Control Message Protocol (ICMP) error message [12]. Figure 1 depicts the sequence of an UDP flooding attack. By permanently addressing different ports, it is ensured that the recipient sends such ICMP error messages. The attack is successful when the volume of malicious packets is such that the victim's bandwidth is consumed and legitimate requests cannot be answered.

E. SYN Flooding

Besides UDP flooding attacks, synchronize (SYN) flooding attacks are the most common type of attack. They exploit TCP, and thus belong to the second category of DDoS attacks. TCP provides reliable, bilateral data transmission between two network-enabled applications. Data is transmitted as TCP segments over the IP. Checksums and the division of data into sequences ensure a complete and error-free transmission. As in the event of an error, individual TCP segments are sent again [13].

A TCP connection is described by two sockets and is uniquely identifiable by its parameters: **socket 1** *source IP address*, *source port* and **socket 2** *destination IP address* and *destination port*. The data to be sent is divided into TCP segments, each of which is assigned a sequence number, to ensure correct data transmission. For this purpose, the window size is defined within the TCP header as a set of data, after which the recipient checks it for completeness and acknowledges it. In addition, various control bits can be set as flags, for example, for the establishment or termination of the connection [13].

Connection setup: A TCP instance with no active connection is in the *LISTEN* state, while waiting for incoming connections. The connection between two TCP instances is established by a *3-way-handshake* (see Figure 2), to synchronize the sequence numbers [13][14]:

1) Host A sends a packet with a random inital sequence number x and the SYN flag set.
2) Host B responds with a set SYN and ACK flag, acknowledges the sequence number of the first message ACK number = $x+1$ and its own random inital sequence number y. A half-open connection now exists. The state of the connection changes on the receiving host from *LISTEN* to *SYN-RECEIVED*.
3) Host A acknowledges the message from Host B with ACK number = $y + 1$ and set ACK flag. The connection between the two hosts is now active and has the state *ESTABLISHED*.

SYN flooding is a DoS attack on a host with a TCP instance. The goal of the attack is to create so many half-open TCP connections on the attacked host that no more legitimate requests can be accepted. For each incoming

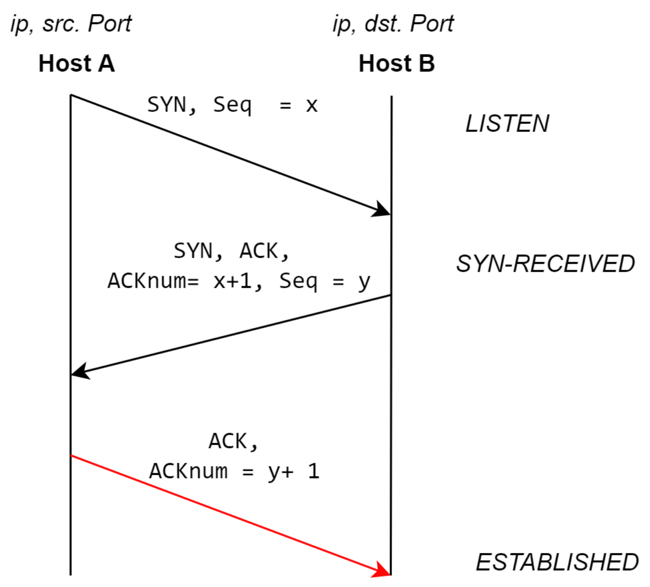

Fig. 2. TCP 3-way-handshake

attacker connection request, a TCP connection must be stored in the backlog queue with its state. The attacker's goal is to overflow this storage structure and prevent legitimate connections from being accepted. To perform the attack, a host opens a TCP connection by sending a synchronize packet. The destination responds with a SYN-ACK packet.

The state of the connection changes from *LISTEN* to *SYN-RECEIVED*. These connections are stored in the backlog, where the connections dwell until the sender acknowledges the receiver's response or the SYN-RECEIVED timer expires. But as the attack hosts are spoofed, they never acknowledge the connection. The attacker is aiming to completely block the backlog. The TCP standard does not define a universal approach for a full backlog. Common implementations now ignore new requests or remove the oldest requests from the backlog. However, an attacker with enough resources can still continuously overfill the backlog.

IV. Methods

To generate the DDoS network traffic, a Python program was implemented that uses the Scapy library, to create network packets. Using Scapy, the packet parameters can be set individually. This allows spoofing of the source IP and source Media-Access-Control (MAC) address. In addition, network packets of different protocols can be generated, to realize the two DoS attacks described above. UDP packets on the one hand, and TCP packets with the SYN flag set, on the other hand.

A. Test Setup

To generate the DDoS network data, a test environment (see Figure 3) was installed with a selection of different IoT devices as attack targets: Amazon Echo 2 and smart cameras from the manufacturers Antela, Nedis, SV3C and Tapo.

These devices are located in a Wi-Fi network created by an Unifi Access Point nanoHD. The Wi-Fi network is integrated into the network created by an Ubiquiti Edge Router via an Unifi Switch Flex Mini. The IoT devices gain access to the Internet via this router.

To record the data traffic, a tcpdump instance runs on the Unifi Access Point nanoHD. This instance intercepts the Wi-Fi internal data traffic via its network interface. The tcpdump recording runs continuously to capture both, normal network traffic from the IoT devices, and attack traffic. To perform attacks on the IoT devices, the attacker connects to the Ubiquiti router.

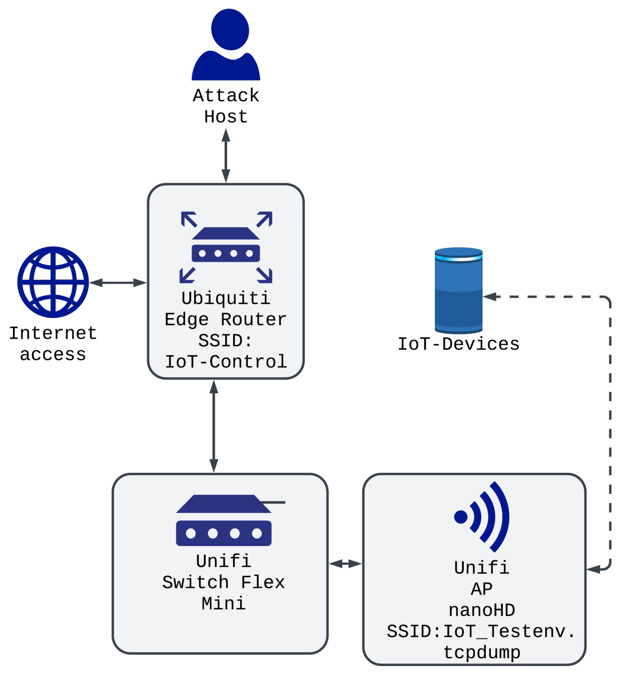
Attack Host
Internet access
Ubiquiti Edge Router SSID: IoT-Control
IoT-Devices
Unifi Switch Flex Mini
Unifi AP nanoHD SSID:IoT_Testenv. tcpdump

Fig. 3. Network diagram of the test environment

B. Implemented Python Program

The objectdiagram of the implemented Python DDoS data generation program can be seen in Figure 4. DDoS attacks are carried out in two stages. In the first stage, the attack is planned. Either manually or randomly generated attacks are planned automatically, for which an attack planner class has been written. In both cases, a target is selected from the IoT test setup, then an attack protocol, either synflooding or updflooding is selected. Next, the attack host network is planned. Each host is identified by a combination of a spoofed IP address, as well as a spoofed MAC address. The number of hosts and the number of packets, to be sent per host, is specified. The planned attack is written to a comma-separated values

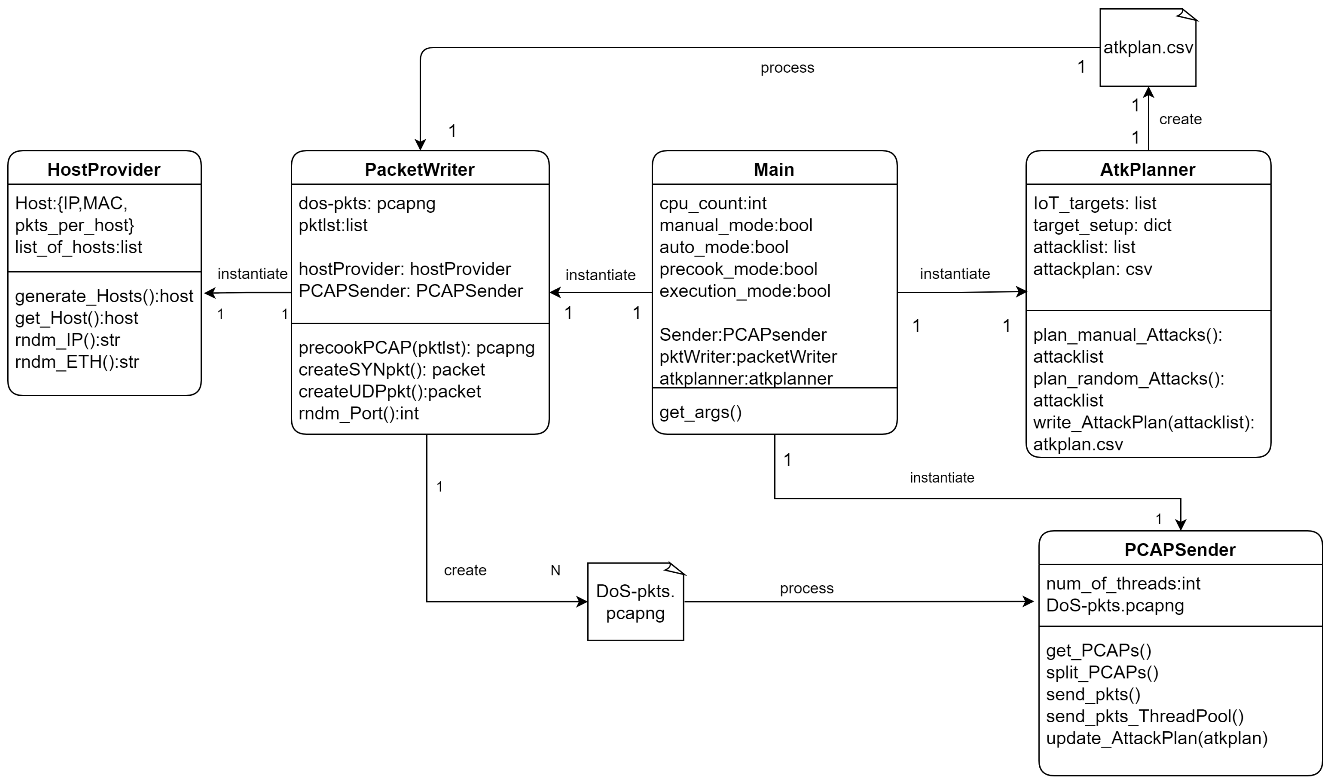

Fig. 4. Objectdiagram of the implemented Python DDoS data generation program

file that serves as persistent storage. After the attack planning is completed, the attack packets are created by a `PacketWriter` class with the help of Scapy and written to a pcapng file.

In the second stage, the actual attack is carried out by the `pcapsender` class. The previously created pcapng files are read into the Random Access Memory (RAM) by Scapy. A threadpool is created depending on the cpu count of the attacker PC. The read packets are divided into chunks for parallelized transmission, depending on the number of cpu cores used. Afterward, the packet chunks are passed to `tcpreplay` by Scapy and processed in parallel.

C. Packet design

On ISO OSI Layer 2, both packet types have a spoofed MAC address. In ISO OSI layer 3, both types of packets contain a spoofed IP address which ensures that no IP addresses from private areas are used. Additionally, this layer contains the destination address of the attack target. ISO OSI Layer 4 is individually designed according to the attack protocol and contains either the UDP or TCP header.

The depicted Python function (see Figure 5) of the `PacketWriter` class takes a host dictionary from the previously created host network, the attack target IP address and its TCP port as parameters. A TCP packet is generated from the transferred data, the spoofed sending

```
def createSYNpacket(self, host,
    target_IP, target_PORT):

     src_MAC = host.get("src_MAC")
     src_IP = host.get("src_IP")
     src_PORT = host.get("src_PORT")

     seqN = random.randint(0, 65535)
     rndm_raw = random.randint(1,1453)
     payload = Raw(b'SYNFLOOD' + b'X'*rndm_raw)

     packet = Ether(src=src_MAC)
     / IP(dst=target_IP, src=src_IP) \
     / TCP(sport=src_PORT, dport=target_PORT,
     flags="S", seq=seqN, window=0)\
     / payload

    return packet
```

Fig. 5. Python function for TCP synchronize packet creation

parameters are taken from fields of the `host dictionary`. A random sequence number is generated for the TCP header. The payload receives a binary coded flag and is provided with a random payload length to increase the variance. Finally, the synchronize flag is set and the window number is defined with 0. The package is assembled layer by layer through Scapy and returned to a packet list.

Analogous to the `createSYNpacket` function (see Figure 5), the Python function `createUDPpacket` creates UDP packets. The spoofed sending parameters are taken from a passed `host dictionary`. The sending port is randomly assigned for more variance. Likewise, the destination port is randomized to ensure that most packets do not reach an UDP application by chance. The payload contains a binary encoded flag to mark the packet as an attack packet in the context of data labeling. In addition, the payload is padded with a random length of binary characters to increase the variance of the packets.

D. Packet sending

For sending, Scapy passes the prepared packages to the `tcpreplay` instance, installed on the attack computer. This creates a temporary pcapng file. This process is the bottleneck of the program, besides the number of threads. However, a loop factor can be defined, so that the same packet is sent several times. This can increase the speed, at the expense of variance. In the case of UDP flooding, this is less serious, since only the payload length represents an increase in the data set. Whereas with the synchronize packets of a synflooding attack, additionally, the same sequence number is sent. This accumulation can also occur in DDoS attacks by real botnets, depending on how the flooding function is implemented. If the loop factor is used, the packet transmission rate increases continuously, since the packets which are read from the temporary pcap file can be stored in the RAM.

E. Data Processing

The records created by tcpdump are exported and processed at cyclic intervals. The network data set can thus be continuously expanded. These pcaps are then read and processed by another Python program. Attack packets are marked as such and labeled with the type of DDoS attack that the data set is suitable for supervised learning procedures. The Python library Scapy is also used for this purpose; its rdpcap function reads in the recorded packets and converts them afterwards into Python dictionaries. Each dictionary is extended by the fields intrusion (boolean) and attacktype (string, *SYNFLOOD* or *UDPFLOOD*), this implementation allows an easy extension with further attack variants in the future. For each packet, it is checked, if the payload contains an intrusion flag. If the test is positive, the intrusion flag is set and the attacktype is noted. If the test is negative, the intrusion flag is set to false and the attacktype is set to none. Optionally, the intrusion flag can be removed from the payload. Otherwise, the payload should not be used for evaluation by machine learning. Finally, the now labeled package dictionaries are inserted into a SQL database for further use.

V. Results

This section describes the results of the generated DDoS data by the implemented Python program. It generates DDoS attack data of the two most common DDoS variants. Continuous data logging in the test environment generates records of idle network traffic intermingled with the network traffic of DDoS attacks. As a result, the records also contain the beginning and ending of the attacks. This is important to train machine learning applications on attack detections.

TABLE I
DDoS attack effects on IoT devices

DDoS effects on IoT Devices		
IoT device	**Attack type**	**Effect**
Amazon Echo	synflood	success
Amazon Echo	udpflood	no success
Nedis Cam Gray	synflood	success
Nedis Cam Gray	udpflood	success
Tapo Cam C100	synflood	success
Tapo Cam C100	udpflood	success
Antela Speed Cam	synflood	success
Antela Speed Cam	udpflood	no success
Nedis cam white	synflood	success
Nedis cam white	udpflood	no success
SV3C camera	synflood	success
SV3C camera	udpflood	no success

The following impact of the attacks on the IoT devices could be observed. The SYN flooding attacks were successful in every case. The devices were unreachable within a few seconds up to a maximum of 60 seconds, for the duration of the attack. Table I depicts the measured effects of the DDoS attacks on the IoT devices.

UDP flooding attacks did not cause every device to fail. This is probably because the IoT devices do not process incoming UDP packets, they ignore them and therefore do not allocate resources.

VI. Discussion

This section discusses the research question defined at the outset. To generate realistic DDoS network data, the attacks carried out must mimic real DDoS botnets. An adequately large host network must be simulated for this purpose. Each of these hosts has an individual combination of spoofed IP and MAC addresses. This means that the network traffic reflects a large number of senders, even though the packets are sent from the same attacking host. This ensures a high variance in the data captured.

The following points address the requirements for the generated data:

1) Packets: Care is taken during packet generation to ensure that valid packets are generated for the protocol in question. This is ensured by evaluating the tcpdump captures. Malformed packets can be detected and corresponding errors have to be fixed, to generate a record of real network traffic.
2) Distribution: To simulate not only DoS attacks, multiple packets are sent simultaneously by using parallelization with threads.

3) Attack duration: Various attack scenarios are simulated. The attack duration takes place in intervals ranging from ten minutes to several hours. Weighing whether to maximize the attack quantity or to follow the parameters as described in the section Anatomy of DDoS Attacks.
4) Non attack traffic: In order to generate qualitative datasets, it is important that the network recordings do not only consist of attacks. So, idle communication of the IoT devices is also recorded.

VII. Conclusion

Finally, we evaluate the advantages and limitations of generating DDoS attack data with the implemented Python program.

A. Advantages

The required type and scope of attacks can be defined individually as part of the attack planning process. This allows data sets to be created according to the user's needs. For example, very long attacks or many attacks, in short succession, can be carried out.

Unlike synthetically generated data, the actual sending of the packets ensures that the network data is correct and error-free. In addition, the network behavior during the attack is authentic. There are fewer potential sources of error in contrast to synthetically generated data, for example, by network simulators. Also in distinction from available DDoS data sets, the data can be generated variably and adjusted according to individual needs.

If a physic attack network were to be implemented for DDoS data generation, this would be accompanied by high material costs for the hardware. In contrast, the implemented Python program is a cost-efficient solution for data generation [15].

B. Limitations of DDoS Python implementation

The DDoS attack program is limited by the number of threads on the attack host. Real DDoS botnets include several thousand hosts. This is not feasible by multithreading with a single attack host. The data transfer rate with the Python program is also lower than in a DDoS attack with a resource-strong botnet.

C. Augmentation of the training data

For the purpose of training data generation, augmentation techniques are used to extend or adapt the existing training data set. The main weakness of the DDoS program is the limited simulatability of a large botnet. Therefore, it is a good idea to first generate a set of realistic DDoS attack datasets as large as possible and then use data manipulation to adjust the training data. Each packet in the network record has a timestamp at which it is captured.

By manipulating the timestamp, multiple attacks that occurred in succession can be combined into a parallel attack to represent a larger attack. This is only limited by the number of spoofed hosts. Augmentation can be done either as part of the package labelling process or retrospectively in the SQL database.

D. iIDS for defending against DDoS attacks

DDoS attacks are difficult to defend against, and existing defenses and prevention measures can be largely mitigated by the attacker. An iIDS is a promising approach.

It usually consists of various modules, which can detect network anomalies based on machine learning techniques and deviations from the usual network traffic of IoT devices. In addition to anomaly detection, the iIDS can classify attacks and distinguish between DDoS attacks and man-in-the-middle attacks [16]. In order to develop an own iIDS, extensive training data is needed. The implemented and described Python program is an approach to generate this training data.

VIII. Outlook

The Python program can be extended to include other DoS attack variants. Scapy allows the variable creation of any network packet. This allows ICMP flooding and HTTP flooding to be implemented. It is planned to split the attacks across multiple physical hosts to create a real distribution, in addition to the simulated distribution through multithreading. For this purpose, several Raspberry Pis are used to perform simultaneous attacks on a target. Subsequently, a larger field trial will be conducted to scale the data generation to a larger scale. Adding more attack hosts to the experimental setup gives a true distribution of the attack. In conjunction with multiple threads, this approaches a larger botnet.

To validate the generated data, it is planned to replicate publicly available DDoS datasets. Similar host quantity and variance will be used, also the packet quantity can be replicated. However, the packet transmission frequency is limited by the hardware on which the Python program runs. For further validation, we plan to create a test setup with an available IDS and train it with our data. This will allow us to compare the results of our system with the available one.

References

[1] Infosecurity Magazine, "DDoS Attacks in 2022: Trends and Obstacles Amid Worldwide Political Crisis," 2022. [Online]. Available: https://www.infosecurity-magazine.com/blogs/ddos-attacks-in-2022-trends/ (visited on 06/07/2023).

[2] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: Classification and state-of-the-art," en, *Computer Networks*, vol. 44, no. 5, pp. 643–666, Apr. 2004, ISSN: 13891286. DOI: 10.1016/j.comnet.2003.10.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128603004250 (visited on 03/01/2023).

[3] N. Sidhu, K. Saluja, M. Sachdeva, and J. Singh, "Ddos attack's simulation using legitimate and attack real data sets," *J. Scientific And Engineering Research*, Jun. 2012.

[4] Research Scholar, SRM University, Sonepat, (Haryana) India., S. Arora*, D. S. Dalal, and Professor, Teerthanker Mahaveer University, Moradabad (U.P), India., "DDoS Attacks Simulation in Cloud Computing Environment," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 414–417, Nov. 2019, ISSN: 22783075. DOI: 10.35940/ijitee.A4163.119119. [Online]. Available: https://www.ijitee.org/portfolio-item/A4163119119/ (visited on 05/10/2023).

[5] A. Balyk, M. Karpinski, A. Naglik, G. Shangytbayeva, and I. Romanets, "Using graphic network simulator 3 for ddos attacks simulation," *International Journal of Computing*, vol. 16, pp. 219–225, Dec. 2017. DOI: 10.47839/ijc.16.4.910.

[6] S. Alzahrani and L. Hong, "Generation of DDoS Attack Dataset for Effective IDS Development and Evaluation," *Journal of Information Security*, vol. 09, no. 04, pp. 225–241, 2018, ISSN: 2153-1234, 2153-1242. DOI: 10.4236/jis.2018.94016. [Online]. Available: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jis.2018.94016 (visited on 03/06/2023).

[7] *Global Journal of Computer Science and Technology*, vol. 14, no. E7, pp. 15–32, 2014, ISSN: 0975-4172. [Online]. Available: https://computerresearch.org/index.php/computer/article/view/100887.

[8] D. Mahajan and M. Sachdeva, "DDoS Attack Prevention and Mitigation Techniques - A Review," *International Journal of Computer Applications*, vol. 67, no. 19, pp. 21–24, 2013. DOI: 10.5120/11504-7221. [Online]. Available: https://www.researchgate.net/publication/258790077_DDoS_Attack_Prevention_and_Mitigation_Techniques_-_A_Review (visited on 06/07/2023).

[9] S. S. Kolahi, K. Treseangrat, and B. A. S. Sarrafpour, "Analysis of udp ddos flood cyber attack and defense mechanisms on web server with linux ubuntu 13," *2015 International Conference on Communications, Signal Processing, and their Applications (ICCSPA'15)*, pp. 1–5, 2015.

[10] A. N. S. Team, *2022 in review: DDoS attack trends and insights*, en-US, Feb. 2023. [Online]. Available: http://www.microsoft.com/en-us/security/blog/2023/02/21/2022-in-review-ddos-attack-trends-and-insights/ (visited on 03/01/2023).

[11] O. Yoachimik, "Cloudflare DDoS threat report for 2022 Q4," *The Cloudflare Blog*, 10.01.2023. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2022-q4/ (visited on 06/07/2023).

[12] Kamaldeep, M. Malik, and M. Dutta, "Contiki-based mitigation of UDP flooding attacks in the Internet of things," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida: IEEE, May 2017, pp. 1296–1300, ISBN: 9781509064717. DOI: 10.1109/CCAA.2017.8229997. [Online]. Available: http://ieeexplore.ieee.org/document/8229997/ (visited on 03/01/2023).

[13] IETF Datatracker, *RFC ft-ietf-tcpm-rfc793bis: Transmission Control Protocol (TCP)*, 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9293 (visited on 06/07/2023).

[14] *Defenses against TCP SYN flooding attacks.* 2006. [Online]. Available: https://www.netconf.co.uk/ipj/ipj_9-4.pdf (visited on 06/07/2023).

[15] S. Alzahrani and L. Hong, "Generation of ddos attack dataset for effective ids development and evaluation," *Journal of Information Security*, vol. 09, pp. 225–241, Jan. 2018. DOI: 10.4236/jis.2018.94016.

[16] J. Graf, K. Neubauer, S. Fischer, and R. Hackenberg, "Architecture of an intelligent intrusion detection system for smart home," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2020, pp. 1–6. DOI: 10.1109/PerComWorkshops48775.2020.9156168.