Automated Network Packet Generation for Evaluation of Neural Networks in Intrusion Prevention Systems

Bernhard Gally
IT Security
FH Technikum Wien
Vienna, Austria
cs19m023@technikum-wien.at

Sebastian Lipp
IT Security
FH Technikum Wien
Vienna, Austria
cs19m032@technikum-wien.at

Damir Marijanovic
IT Security
FH Technikum Wien
Vienna, Austria
cs19m031@technikum-wien.at

***Abstract*—This paper is a part of a project. It describes the part of automated network packet generation. The generated packets are used to evaluate neural networks in intrusion prevention systems. The challenges in this project include building the infrastructure and connecting individual nodes in the network.**

***Index Terms*—artificial intelligence, neural networks, computer networks, network security, packet generation**

I. Introduction

For a good, respectively successful detection of malicious network traffic using neural networks, the neural network has to be trained well. Therefore a large set of labelled network traffic is necessary. The more attack scenarios there are, the better the differentiation between attacks and normal traffic. Of course, not all possible scenarios can be dealt with here, which is why we have to concentrate on the attacks that occur most often in the real world. This subproject relates to the generation, labelling, distribution and recording of network data and packets.

The network has to be deliberately kept simple and the different types of attacks have to be treated separately. The infrastructure is considered from scratch and implemented into a network simulation tool called *GNS3*. To make the individual parts dynamic, docker containers are used. This makes it possible to evaluate the individual scenarios separately. The traffic is recorded using the *tcpdump*, and the entire filtered traffic is saved in so-called pcap files. As far as the hardware is concerned, sufficient computing power is required, as well as sufficient storage space to be able to record the traffic.

The design of these parts are covered in the next two sections. The first part describes the infrastructure to create simulation environments. The second part focuses on the implementation of specific scenarios.

II. Infrastructure

This chapter covers the design of an infrastructure which provides a tool for creating generic environments in which specific network scenarios can be simulated to record network packets and extract features with which neural networks inside intrusion prevention systems could be trained and evaluated.

This work is split into four parts. The first part designs the fundamental simulation architecture. The second focuses on the creation of network nodes, followed by the configuration of the network in the third part. The last part covers the automation of the procedure with the help of provioning tools.

A. *Designing the architecture*

A simulation network consists of multiple scenarios based on network attacks and benign traffic. In this network, generic hosts can be placed and connected to each other or the internet with the usage of hubs, switches and routers. These hosts reflect attackers and victims or usual clients and servers, as well as recorders. To ensure recorded features can be labeled, recorders are placed on all positions where another kind of attack or network traffic should be recorded. Furthermore, they are able to filter out selected traffic to ensure only the important traffic is recorded. This means network traffic is recorded within multiple files marked as bad or good which serve as input for the data extractor to extract the features and create the dataset to train and test the neural network.

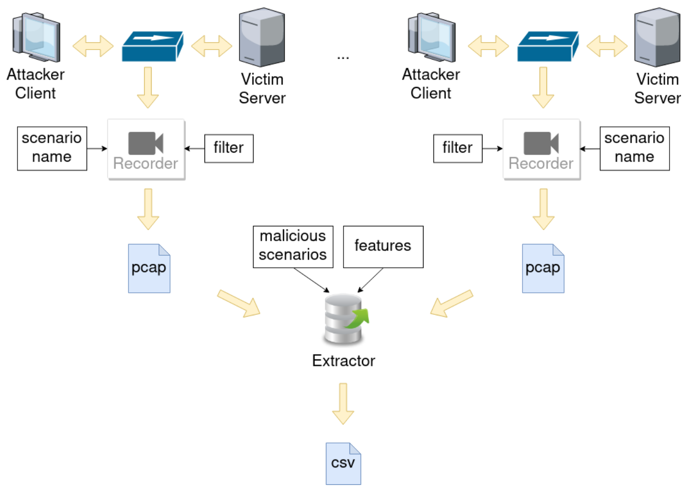

Fig. 1. Design principle

Fig. 1 shows the design principle. The recorders receive the data through network hubs placed between the attackers and victims or clients and servers and record in pcap files to a central filesystem. The scenario names and filter settings are passed to the recorders as parameters. The recorded files - named after the associated scenario - are then collected and forwarded to the extractor which knows the names of the malicious scenarios and extracts the selected features to a csv file. The extractor is only required once and is not part of the simulation environment.

B. Creating the nodes

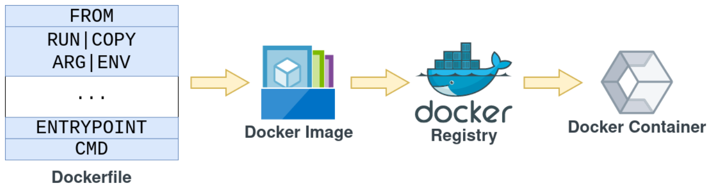

Fig. 2. Node creation

Docker containers are used for creating customized network nodes to offer users a very generic approach. The configuration of the nodes is based on code and can be easily created. It is defined within Dockerfiles which can be transformed to an image and uploaded to a registry through Docker client tools as shown in Fig. 2. When the images are placed on a public Docker registry, they can be pulled and executed from anywhere. This is the default approach in the current infrastructure design.

The lines in a Dockerfile describe the layers of which the image consists. The first line is a statement to specify the base image, for example Alpine Linux or Ubuntu. The new image inherites everything from this base layer. After that, multiple commands can be added to install software, configure the system, copy files to the image or define environment variables. The last lines define the entrypoint and/or a command which the container runs when it is started. This could be for instance a program to attack a host or record the network traffic. Docker also supports mounting volumes to the containers when they are started as well as passing arguments to use the same image for similar nodes with different settings. More about creating Dockerfiles can be found on the offical documentation of Docker.[1]

Once the images are built and uploaded to an accessible registry, they can be further used in the simulation environment. The target hosts only need the Docker runtime environment installed and access to the image registry. In this case the official Docker Hub is used to build and store the images for the scenarios. This includes all clients, servers, the recorder and the extractor.

The network recorder is based on a customized Docker image which runs tcpdump with additional arguments like the filename to specify the scenario name and the filter expression to sort out unwanted traffic. It records the packets passing the first interface. For more information about possible commands and configuration refer to the manpage of tcpdump.[2]

The extractor, or more precisely the CI flowmeter, is also placed into a Docker container mounting a volume containing all the recorded pcap files. It extracts only the features defined by a parameter and saves the csv file with the extracted features at the same place as the pcap files, which can be later transferred to another host to evaluate the data.

The runtime configuration and connection of the network nodes as well as the creation of other nodes like switches, hubs and routers is part of the simulation environment which is covered in the next section.

C. Connecting the nodes

The configuration of the network, or more precisely the connection between the network nodes, is done within GNS3, an open-source network simulator which can be easily controlled via a native graphical user interface or RESTful API. It consists of a server and a client application. In this setup, the server application is installed on a virtual machine which offers 8-cores, 32 GiB RAM and 300 GiB hard disk space. It is possible to install multiple servers, but in this case one was sufficient for simulating multiple scenarios including DoS attacks. More information about the installation of the server can be found on the official website of the GNS3 project.[3]

Once the server runs, a project can be created and the scenarios can be configured via the graphical interface of the GNS3 GUI. It is recommended to setup a VPN to the server to ensure the terminals of the network nodes are reachable remotely without configuring network tunnels. This means it is necessary to connect to the server via VPN before the GUI can reach the server applicaiton.

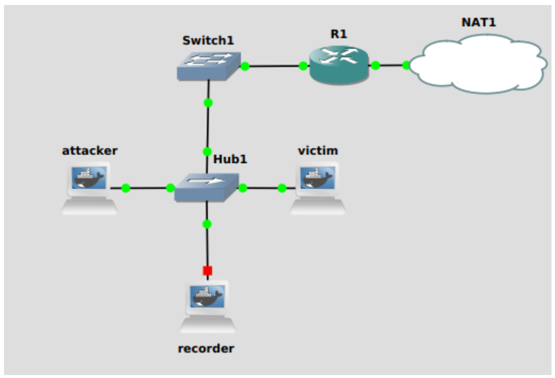

Fig. 3. Network example

Fig. 3 shows a simple network created in GNS3. The attacker, victim and recorder are created through Docker templates referring to the prevously created Docker images. GNS3 supports the configuration of environment variables and volumes for Docker containers. All other components,

[1]https://docs.docker.com/develop/develop-images/dockerfile_best-practices

[2]https://www.tcpdump.org/manpages/tcpdump.1.html

[3]https://docs.gns3.com/1c2Iyiczy6efnv-TS_4Hc7p11gn03-ytz9ukgwFfckDk/index.html

including switches, routers and the interface to the internet, are basic network components of GNS3 and are already available in the device browser. Cisco routers need official IOS images before they can be created.

The project is automatically saved to the filesystem and includes all components which are needed to run the network, including configuration, logs and recorded data. The simulation can be started through the GUI or the API and produces the pcap files within the project directory. Once the simulation is stopped, the pcap files can be copied to a temporary directory and sent through the extractor container to produce the csv file including the extracted features.

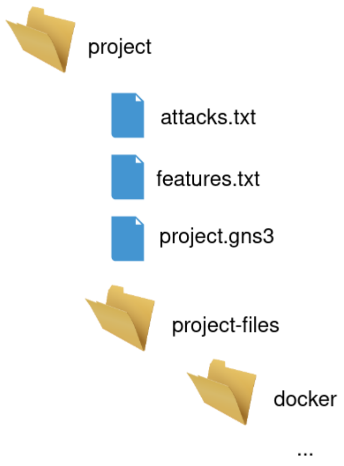

Fig. 4. Project structure

The setup of the network is done through the GUI because the API does not support all features to create network nodes. The project folder can be checked into version control and copied to the server when it is needed. As shown in Fig. 4, it consists of the gns3 file which contains all the network configuration in a JSON format. The project-files directory contains additional files which are part of the filesystems of the network nodes, for example the docker containers. The text files are the parameters, which are passed to the extractor to determine the attack names and the features to extract.

D. Automating the procedure

The automated setup of the server and run of the simulation is done within Anisble playbooks. Therefore, multiple roles are created, one for the setup of the GNS3 server, one for running the network simulation and one for extracting the features. As shown in Fig. 5, the server role is associated with the setup playbook, while the simulation and extraction roles are used by the run playbook. These playbooks are stored in YAMl files and can be executed with the ansible tools.[4]

The server role sets up the GNS3 server and OpenVPN, copies the device images and projects and configures the Docker runtime environment. It downloads the VPN configuration to the client, which can be later used to connect to the

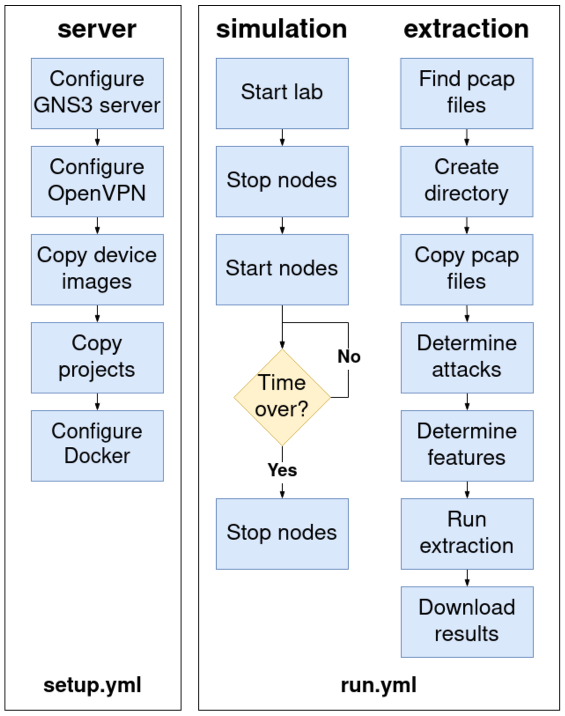

Fig. 5. Ansible roles

server. Because the test setup restricted the incoming ports, VPN was changed to use TCP instead of UDP to enable the functionality of creating SSH tunnels.

To run the simulation role, a VPN connection to the server is required. This is accomplished by a simple connection script which opens an SSH tunnel and connects to the server via VPN. This is necessary because Ansible needs access to the GNS3 API and the network components. When the role is executed, it opens the project and restarts all network nodes. The duration can be defined through a parameter. When the time is over, the nodes are stopped. To control GNS3 via the API, an additional Ansible collection is used. [5]

The extraction role gathers the pcap files, creates a temporary directory and copies all files there. Then it determines all attack scenarios and features from a file in the project directory and runs the extraction container with the mounted pcap volume and the passed parameters. After that it downloads the csv file with the extracted features. If specified in a parameter, the pcap files are also downloaded.

The server can be setup by running the following command:

```
ansible-playbook -i hosts setup.yml
                 --ask-become-pass
```

[4]https://docs.ansible.com/ansible/latest/user_guide/playbooks.html#working-with-playbooks

[5]https://github.com/davidban77/ansible-collection-gns3

Once the GNS3 projects are checked into the Ansible project folder, the simulation can be started with the following command:

```
./connect.sh
ansible-playbook -i hosts run.yml
                 --ask-become-pass
                 -e project=<project_name>
                 -e duration=<minutes>
                 [-e pcap=true]
```

All Ansible commands must be executed within the gns3 folder and need the sudo password from the server user. The hosts file contains the target hosts configuration.

The next chapter focuses on the implementation and simulation of specific scenarios within a GNS3 project with the help of the introduced infrastructure to produce a new dataset for evaluating neural networks in IPS systems.

III. Simulation

Benign and malicious traffic have to be generated for multiple scenarios in order to provide a diverse dataset that can be used to train the neural network. As previously mentioned these scenarios are divided into separate groups, each with their own recorder so that benign and malicious traffic do not mix. Each attack and its benign counterpart were implemented using Docker containers which can be found on Dockerhub.

A. Secure Shell (SSH)

SSH is one of the most common protocols used today. By utilizing it secure connections over the internet can be established to remote servers in order to execute commands or copy files from one machine to another. Simulating such a configuration therefore requires a server, a benign client and a malicious client

1) SSH server: To allow a client to connect to the SSH Server, the client has to exist on the remote host. We therefore added a new user with a known password to the server. Because the password can only be set in an interactive way via useradd, we had to use the chpasswd utility. The configuration of the server can be seen below:

```
FROM ubuntu:latest

RUN apt-get update && \
    apt-get install -y openssh-server && \
    mkdir -p /var/run/sshd && \
    useradd technikum -m -d /home/technikum && \
    echo technikum:technikum | chpasswd

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

2) Benign client: The idea behind our implementation of benign SSH traffic is to first connect to an SSH server. Secondly, send commands that are executed on executed on the remote server which return a response message that generates traffic over the network. Because each command will only produce a minute amount of traffic multiple clients are needed to increase the data volume. Furthermore, since it is not possible to enter a password via the default ssh tool in a non-interactive way sshpass had to be installed. This utility allows us to provide the password used for the SSH login in a non-interactively. The exact implementation of the benign traffic container that was used in our simulation can be seen below:

```
FROM ubuntu:latest

ENV targetip=192.168.1.100
ENV command="echo test; mkdir test; rmdir test;"

RUN apt-get update && \
    apt-get install -y sshpass

ENTRYPOINT  while true; \
            do sshpass -p "technikum" \
            ssh -o StrictHostKeyChecking=no \
            "technikum@${targetip}" ${command}; \
            sleep 5; done
```

3) Malicious client: By making use of a password list an SSH brute-force attack can be executed quite easily on an SSH server. An attacker uses the list in combination with a given username to attempt a successful login on the victim machine. Luckily, some Docker images created for pen-testing come equipped with such password lists already. The utility that was used to attack the SSH server is called hydra, a parallelized password cracking tool which supports multiple protocols and the use of password lists. To further increase the generated data multiple containers were used. The configuration of the malicious ssh containers can be seen below:

```
FROM c4nc/kalibase:latest

RUN gunzip /usr/share/wordlists/rockyou.txt.gz

ENTRYPOINT ["hydra", "-l", "root", \
            "-P", \
            "/usr/share/wordlists/rockyou.txt", \
            "192.168.0.10", "-t", "4", "ssh"]
```

B. Botnet

The second scenario that was used is a botnet. In a botnet a master or server controls multiple agents to perform some action. Screenshots, file downloads, uploads, installations and much more can be done once the server has control over the infected pc.

1) Server: The server controls the agents with a list of commands. The server send these commands out to the agents which then perform the requested operations. In our simulation we let all clients do the following operations repeatedly:

```
apt -y install tor
download \
  http://ipv4.download.thinkbroadband.com/5MB.zip
screenshot
mkdir -p /tmp/zzz
cp /etc/passwd /tmp/zzz/
cp /etc/shadow /tmp/zzz/
zip zzz.zip /tmp/zzz/
upload zzz.zip
rm -rf /tmp/zzz zzz.zip 5MB.zip
apt -y remove tor
download \
  http://ipv4.download.thinkbroadband.com/100MB.zip
apt -y install curl wget netcat
```

2) Agent: Similarly to the benign traffic, a single host would not generate a noticeable amount of network traffic. Multiple agents were added to the scenario to increase the generated data. By using Ares the required configuration for the agents is the port and IP address of the control server:

```
FROM ubuntu:18.04
USER root
WORKDIR /opt

RUN apt-get update && \
    apt-get install -y \
      git \
      python-pip && \
    apt-get clean all

RUN git clone \
    https://github.com/sweetsoftware/Ares.git && \
    cd Ares && \
    pip install -r requirements.txt

ARG SERVER=192.168.0.200
ARG PORT=8080
WORKDIR /opt/Ares/agent
RUN sed -i "s/localhost/$SERVER/g" ./config.py
RUN sed -i "s/8080/$PORT/g" ./config.py
ENTRYPOINT ["./agent.py"]
```

C. Goldeneye

Goldeneye is a DoS attack that can take down webservers. Similar to other DoS tools like Slowloris it occupies all available sessions of the webserver and therefore locks out anyone else trying to connect to it. Unlike Slowloris it uses Keep-Alive to keep its connected sessions alive. In order to be able to use this attack a webserver must be provided. Because of performance issues the attack could not be ran continuously and had to be timed out every few minutes.

D. Benign traffic

Network traffic consists of many different network protocols and connections. Furthermore, attacks generally speaking are only a small portion of the network traffic. Most of the traffic is benign traffic that also has to be simulated in some way in our environment. To overcome this problem multiple benign containers were started that generated harmless traffic. Downloading files and browsing the internet are the most common type of traffic in normal network. Although it is worth mentioning that the definition of normal traffic is highly dependant on what kind of servers are connected to the network. A server farm for example has a different definition of normal traffic than a home network regarding amount and protocols used. For our environment we considered file downloads and web browsing as normal traffic. Hence, Docker containers that downloaded 10MB, 100MB files were added to the network. A third container was added to browse the web. The containers do not store the downloaded files anywhere. They are redirected to /dev/null meaning the containers do not need any significant amount of storage space but still download every byte of the requested file before restarting. After the downloads are finished the container waits 5 seconds before it retries the same operation. The configurations of these containers can be seen below:

```
FROM ubuntu:latest

ENV website="google.at"

RUN apt-get update && \
    apt-get install -y wget

ENTRYPOINT  while true; \
            do wget -O /dev/null ${website}; \
            sleep 5; done
```

IV. Conclusion

The result of this project can be used in the knowledge as well as in the structure. Due to the dynamic structure of the infrastructure, extensions can be added with little effort. During the implementation of the infrastructure, minor difficulties came up again and again, but these could usually be remedied with a relatively small change. Among other things, difficulties arose in the transfer of environment variables in the individual Docker containers. The other part of this project used this project as the basis, in which the recorded information and packages were evaluated. The successful evaluation showed that both the infrastructure and the individual nodes worked well.