# A DISTRIBUTED VIRTUAL COMPUTER SECURITY LAB

Harald Vranken
*Open Universiteit, School of Computer Science, P.O. Box 2960, 6401 DL Heerlen, The Netherlands*

Jens Haag, Tobias Horsmann, Stefan Karsch
*Cologne University of Applied Sciences, Faculty of Computer Science and Engineering Science*
*Steinmüllerallee 1, 51643 Gummersbach, Germany*

*Keywords:* Distance education, e-Learning, Distributed virtual computer security lab, Virtual lab.

Abstract: Universities offering courses in the field of analyzing, configuring and safeguarding computer networks, provide specific software and hardware resources to students for practical assignments. At distance universities these resources usually are not physically accessible for remote students. We initially addressed this issue by offering an environment which allows students to build virtual computer networks on their local computer. The environment consists of a preconfigured virtualized software package and is referred to as the virtual lab. This approach reaches its limits when students intend to perform group work similar to typical on-site courses. To remove this limitation, we developed an extended virtual lab, called the Distributed Virtual Computer Security Lab (DVCSL), in which distinct remote virtual labs can be connected across a connection network (e.g., the internet). The DVCSL allows remote students to perform networking and security exercises inside an encapsulated distributed common networking environment. The design of the DVCSL meets two major requirements: establishing a transparent communication path between remote virtual labs and assuring that non-participating systems outside the DVCSL are not affected by the transmitted data. In this paper we present the architecture of the DVCSL and demonstrate its functionality as well as its security by an example setup.

## 1 INTRODUCTION

Computer security labs are of great value in courses that teach security of computer systems and networks. The knowledge that students learn from textbooks, is illustrated, deepened, and anchored by carrying out practical exercises in such a lab. In addition, students usually like carrying out practical exercises next to studying theory, thereby improving their motivation and results.

Computer security labs are nowadays quite common at many universities; see for instance (Gaspar, Langevin and Armitage, 2007) and (Mattord and Whitman, 2004) for an overview of some recent practices. Initially, a computer security lab was a room containing computer systems connected in a network, completely isolated from the outside world. Administration and maintenance of such a lab however is labor-intensive. Students work in the lab with super-user rights and can modify system configurations at will. After a session, it is necessary to clean up system configurations, which may even require reinstalling operating systems. Therefore, most modern computer security labs apply virtualization, where students work in virtualized environments. Cleaning up or reinstalling a virtual lab simply means reloading the virtual environments, which can even be an automated task. Early examples of labs applying virtualization have been reported in (Bullers, Burd and Seazzu, 2006), (Hay and Nance, 2006) and (O'Leary, 2006).

Isolated computer security labs at universities are however not suited for distance teaching, since students may not be able to travel to the lab due to restrictions on time or distance. Distance teaching is accommodated by providing remote access to the lab over secure network connections. Numerous examples of computer security labs with remote access have been reported, such as in (Border, 2007), (Hu, Cordel and Meinel, 2005), (Keller and


Vranken H., Haag J., Horsmann T. and Karsch S..
A DISTRIBUTED VIRTUAL COMPUTER SECURITY LAB.
DOI: 10.5220/0003343801100119
In *Proceedings of the 3rd International Conference on Computer Supported Education* (CSEDU-2011), pages 110-119
ISBN: 978-989-8425-49-2

Naues, 2006), (Krishna, Sun, Rana, Li and Sekar, 2005), and (Lahoud and Tang, 2006). Although such labs may be accessed remotely at any time from any place, they are generally not easily scalable. Allowing an arbitrary number of students to participate at the same time, requires students to reserve timeslots in advance for working in the lab. This may impose restrictions for students in distance education, who usually study in evening hours and weekends. Provisioning a remote lab for peak access outside office hours, may result in a largely over-dimensioned lab with a low average degree of utilization and hence a waste of resources.

Some universities recently adopted a different approach for providing a virtual computer security lab (VCSL), for instance as reported in (Vranken and Koppelman, 2009) and (Li, 2009). Instead of moving students to the lab, either physically or by remote access, the lab is moved to the students. The VCSL of the Open University of The Netherlands (Vranken and Koppelman, 2009) consists of an isolated, secured software environment, provided on a DVD, that each individual student can easily install on his/her computer. Inside the VCSL, the student can set up multiple virtual hosts, connect them into virtual computer networks, and safely carry out experiments related to security. The VCSL is used in courses on computer security, but could also be used in courses on operating systems, computer networks, distributed systems, or web services. In the VCSL, the student can work both as a hacker, preparing and launching attacks against systems and networks inside the VCSL, as a system administrator, implementing security measures to prevent and detect attacks, and as a regular user, experiencing the effects of attacks and security measures.

The VCSL can be easily installed and run locally on each student's computer. This decentralized approach is suited to accommodate any number of students, provides students the freedom to run the lab whenever and wherever they want, and eliminates the need for a central lab at the university. A shortcoming however is that students have to work on their own. It is for instance impossible to offer exercises on distributed attacks involving large botnets, or hacking games in which students are challenged to attack each other's systems and securing their systems against attacks from fellow students. We therefore extended the VCSL towards a distributed virtual computer security lab (DVCSL). The DVCSL allows connecting the VCSL's of multiple students into a large virtual network running over the internet. Traffic inside the virtual network is completely isolated from the outside world. Hence, communication between virtual hosts inside the DVCSL and hosts outside the DVCSL is impossible. Students can therefore safely carry out assignments related to security without any restrictions - even spreading malware could be allowed - without the risk of accidentally (or intentionally) attacking or infecting hosts on the internet.

This paper presents the communication infrastructure of the DVCSL. The paper is organized as follows. In section 2 we give further details on the VCSL, the basic building stone of the DVCSL. In section 3 we list related work on computer security labs, showing the novelty of our DVCSL. In section 4 we explain the architecture and implementation of the DVCSL. In section 5 we describe an example setup of the DVCSL, demonstrating the correct, secure operation of the DVCSL. In section 6 we shortly address future work and section 7 concludes the paper.

## 2 VCSL

The VCSL is a stand-alone environment, composed of two nested software virtualization layers, that each student can install on his/her computer. The software components to build the VCSL are freeware or open source, and are distributed to students on a DVD.

The VCSL is composed of two virtualization layers, as shown in figure 1. The host machine is the student's computer, which runs an arbitrary operating system, i.e., the host operating system. The first virtualization layer creates the virtual host machine. It consists of virtualization software such as VMware Player (freeware) or Oracle VM VirtualBox (open source), which runs on the host machine just like an ordinary application. Versions of this software are available for a large range of platforms. VirtualBox for instance runs on host machines with either Windows, Linux, Mac OS X, or OpenSolaris. This first virtualization layer therefore runs on nearly all student computers, regardless of the hardware and the host operating system. The virtual host machine runs the virtual host operating system. For the VCSL we selected Linux, since it is open source and can be distributed to students without licensing costs. In fact, we selected Knoppix, a bootable live Linux system containing a collection of GNU/Linux applications and the KDE graphical desktop environment.

The second virtualization layer is a Linux application, called Netkit (Pizzonia and Rimondini, 2008), that runs inside the virtual host machine. This second virtualization layer allows to instantiate

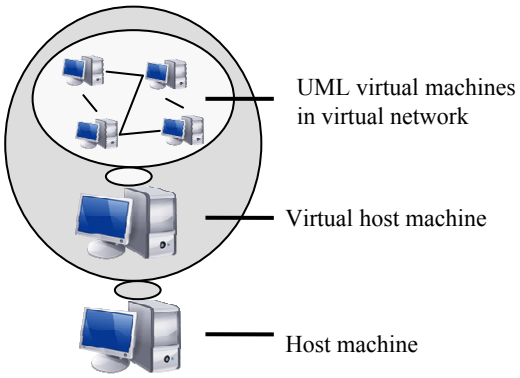


Figure 1: Architecture of VCSL.

multiple virtual machines that all run Linux. Netkit applies virtualization based upon User Mode Linux (UML). A UML virtual machine is created by running a Linux kernel as a user process in the virtual host machine (Dike, 2006). Multiple UML virtual machines can be run easily with minimal resources. The Linux file system is shared by all UML virtual machines using the copy-on-write (COW) mechanism. Hence, the file system is shared read-only by all UML virtual machines. Each UML virtual machine has a second, separate file system in which only the local changes to the shared file system are stored. This saves both disk space and memory, and simplifies management of multiple UML virtual machines. Netkit consists of a collection of scripts that allow to setup and configure UML virtual machines with virtual network interfaces, and to connect these into virtual networks. The virtual network interfaces of the UML virtual machines can also be connected to the virtual network interface of the virtual host machine, and even to the physical network interface of the host machine.

The hardware requirements for running the VCSL are very modest. A few UML virtual machines can already be run smoothly on a PC with a Pentium-4 processor and 256 MB memory. The VCSL has been used successfully in a security course by more than two hundred students with only few minor problems.

## 3 RELATED WORK

The idea of using an isolated network as anenvironment to perform security related tasks for the purpose of research or education is evident (Bishop and Heberlein, 1996). There are two general approaches to create such an environment.

The first approach is to create or use an isolated, physical network with physical hosts that is separated from a productively used network such as a campus network (Bishop and Heberlein, 1996; Jakab, Janitor and Nagy, 2009). This isolation may be achieved by physical separation of the networks or by using components like firewalls to restrict data flow between network areas (Yang, Kwok-Bun, Liaw, Collins, Venkatraman, Achar and Sadasivam, 2004). Within this isolated network the students can perform exercises and work with a real-world like network setup. Remote access to such a network may be granted by using remote access technologies such as VPN (Virtual Private Network).

The second approach makes use of virtualization technologies to create an isolated, virtual network with virtual hosts. Literature refers to such an environment in the context of education or e-learning usually as a virtual lab (Damiani, Frati and Rebeccani, 2006; Keller and Naues, 2006). This approach significantly reduces the amount of physical hardware resources (e.g., switches, routers, hosts), since the required resources are created by virtualization.

Literature also reports two main approaches to provide an isolated network. In the first one, the environment is provided by a central authority, usually located at the university (Drigas, Vrettaros, Koukianakis and Glentzes, 2005) and students can get physical or remote access by using a secured network connection. Second, the environment is provided as a preconfigured, stand-alone software package which can be installed and used by students on any computer, usually their private computer (Li, 2009; Vranken and Koppelmann, 2009). This gives the students the opportunity to safely carry out assignments wherever and whenever they want to. As flexible and independent this kind of environment is, it lacks the option to connect several of these stand-alone environments. It is not possible for remote students to cooperate in teams when carrying out exercises. This also means that the demand for real world-like scenarios that require to work with or against other students, is sometimes not fulfilled.

In this paper, a first step is introduced to connect two or more local, virtualized, stand-alone environments to create a distributed virtual computer security lab.

## 4 DVCSL

In this section we outline the architecture and implementation of the DVCSL. Each VCSL consists of a number of UML virtual machines, connected by

virtual networks, and also connected to the virtual host machine. For building a DVCSL, a transparent connection between the virtual host machines of distinct VCSL's is required. We therefore equip each VCSL with an interface, such that a point-to-point connection can be created between two VCSL's. We will outline this further in the remainder of this paper. A future extension is to connect multiple VCSL's, which is achieved by adding a central authority that manages all connected VCSL's and forwards data between them. The latter is subject of our ongoing research and is not included in this paper.

### 4.1 Requirements

In order to connect two VCSL's transparently, we connect the VCSL's at OSI-layer 2. We presume Ethernet-based networks, and hence the Ethernet protocol running at OSI-layer 2. Connecting two virtual networks at OSI-layer 2 requires that Ethernet frames in the first virtual network should be transmitted transparently to the second, remote virtual network and vice versa. By connecting VCSL's at OSI-layer 2, the connected virtual networks will behave like a single broadcast domain (Comer, 2001). Hence, students working in the DVCSL have the notion of being connected to other students over an Ethernet LAN, although the actual connection is by a WAN using the public internet involving the entire TCP/IP protocol-stack. This is in contrast with conventional network operation, where LAN-frames are converted into WAN-packets at the network perimeter by gateways, preserving the original payload of the upper OSI layers. Our approach also differs from VPN's, where tunneling is generally done at OSI-layer 3 (e.g., when applying IPsec) or above (e.g., when applying SSL).

Performing practical exercises on networking and IT-security introduces an additional requirement: network data that is transported inside a DVCSL must not harm non-participating systems also connected the WAN. It should be absolutely assured that non-participating systems, such as the host systems on which the VCSL's run and any other hosts in the internet, are not affected by the transmitted data. Hence, traffic inside the DVCSL should be completely isolated from the world outside the DVCSL.

To connect virtual networks, we first examine the virtual networking architecture of Netkit. Netkit consists of a collection of scripts which configure and deploy UML. As mentioned, UML is a Linux-kernel executed as a user application. UML can run its own processes inside its kernel environment. An UML-kernel does not interact directly with the hardware, but with the system call interface of the underlying Linux-kernel (Rimondini, 2007), which we referred to as the virtual host operating system in figure 1. Netkit allows to easily configure and set-up UML virtual machines, and building and using local virtual networks. Virtual networks are created by the so-called UML-Switch, a tool providing network logic that comes with the UML-Utilities tools in UML.

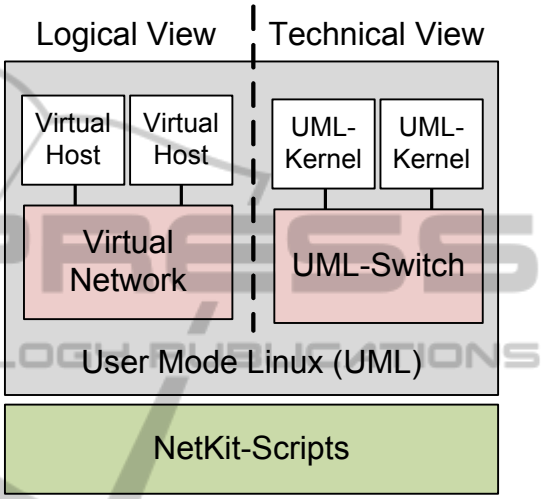


Figure 2: Architecture of Netkit.

Figure 2 shows the architecture of Netkit and provides a logical and a technical view of its components. The logical view shows two virtual hosts which are connected in a virtual network. In the technical view, the virtual hosts are UML-kernels that communicate with each other using the service provided by a UML-Switch.

As the name implies, a UML-Switch connects two or more virtual hosts with switch-like behavior. A network switch establishes a logical point-to-point connection between hosts in a network that are connected to the ports of the switch.

The UML-Switch can also be configured to behave like a hub (Dike, 2006). Virtual hosts that are connected in a virtual network with hub-like behavior can be considered to be connected in the same network segment (Schreiner, 2009). In a network segment, a host receives all data sent by other hosts in the network segment, even if the host is not the designated receiver of the data. Netkit uses the UML-Switch with this hub-like behavior, which offers users the opportunity to analyze data transmitted from any virtual host in the virtual network.

In the DVCSL, we c onnect two remote virtual

networks by extracting data at a local UML-Switch and sending it across a connection network to a remote UML-Switch where the extracted data is then injected, and vice versa. In this way two virtual networks can be connected. Implementing this approach is done by two subtasks:

- Subtask A: extracting and injecting network data at a UML-Switch.
- Subtask B: sending and receiving the extracted network data across a network.

## 4.2 Subtask A: Extracting and Injecting

We examine the technical implementation of a virtual network as shown in figure 2 in which the UML-Switch is involved. For each virtual network, an instance of the UML-Switch is running, which in fact uses a UNIX-Socket to communicate with the virtual hosts. A UNIX-Socket is a system resource which serves as a communication endpoint and can be used for remote communication or local inter-process communication (Stevens, Fenner and Rudoff, 2006).

The UML-Switch attaches itself to a UNIX-Socket, and listens for incoming data sent by the virtual hosts to the UNIX-Socket. The UML-Switch reads the network data from the UNIX-Socket and writes the data to all other connected virtual hosts, which creates the hub-like behavior of the UML-Switch.

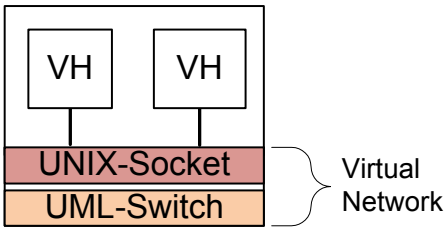


Figure 3: Construction of a virtual network.

Figure 3 shows the architecture of a virtual network. For simplicity, only two virtual hosts (VH's) are shown, but multiple VH's can be connected to the virtual network. The VH's are connected to the UNIX-Socket. Communication over the virtual network is realized by the VH's sending data to the UNIX-Socket, and the UML-Switch forwarding this data to all other connected VH's in the same virtual network.

Extracting data from the UML-Switch and injecting data from a remote virtual network into the UML-Switch, relies on the hub-like behavior of the UML-Switch. We developed a new software component, that is applied in the virtual host operating system and that connects itself to the UNIX-Socket of the virtual network. This component can be considered as a ghost host in the virtual network because it is completely transparent to the other components. With this ghost host it is possible to extract all Ethernet frames from the UNIX-Socket, as well as to inject frames received from a remote virtual network. Compared to a normal virtual host, the ghost host is not jailed in the Netkit environment. The ghost host can therefore communicate with the outside world, which cannot be done by a normal virtual host. The UML-Switch with hub-like behavior guarantees that network data received from a remote virtual network is distributed to all other locally connected virtual hosts.

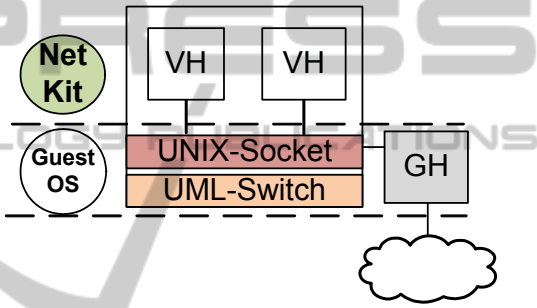


Figure 4: Virtual network with ghost host.

Figure 4 shows the construction of the virtual network in figure 3. The figure also shows the logical layers of the virtual network. The virtual hosts are located and jailed within the Netkit layer. The implementation of the virtual network is realized in the virtual host operating system, which may also be referred to as the guest OS. In the guest OS layer a ghost host (GH) is located which realizes subtask A. The ghost host can extract all data that is sent over the virtual network and can forward the data to a destination outside the guest OS (cloud). Incoming data from outside (cloud) can be injected in the UNIX-Socket using the ghost host.

Hence, the ghost host provides an interface to the virtual network, where Ethernet frames can be extracted and injected without the need to modify any existing components of Netkit or UML.

## 4.3 Subtask B: Sending and Receiving

Subtask B consists of sending the extracted Ethernet frames from one ghost host to another ghost host in a remote system. Ethernet frames cannot be sent directly over the internet or a WAN (Comer, 2001),

since those networks require a network protocol which runs on OSI-layer 3, presumably the Internet Protocol (IP). To resolve this issue, the Ethernet frames are sent over a remote bridge which is established between two ghost hosts.

A remote bridge can connect two distant networks by using a connection network (Schürmann, 2004). A benefit of the remote bridge is that the connection network and the distant networks can use different protocols. Even if the protocols of the connection network and the distant networks are equal, the remote bridge ensures that the distant networks cannot intercommunicate with the connection network. The remote bridge consists of two remote bridge endpoints which connect two networks at OSI-layer 2. Such an endpoint encapsulates an Ethernet frame of the local environment in a transport protocol, and sends the data to the remote endpoint where the transport protocol is removed and the OSI-layer 2 data is injected in the remote environment.

We extended the ghost host component and added functionality of a remote bridge endpoint. The Ethernet frames that are extracted by the ghost host, are first encapsulated in the IP protocol (acting as transport protocol), and next sent to the remote bridge endpoint of a fixed distant destination. For incoming data, the IP protocol is removed by the ghost host and the Ethernet frames are sent into the local network, respectively the local UNIX-Socket.

### 4.4 Building the DVCSL

With subtask A and B in place, we can connect two distant virtual networks by interfacing to a remote bridge between the virtual networks. Extracted data of one virtual network is encapsulated by the remote bridge endpoint into a transport protocol, and sent to a distant remote bridge endpoint where the data is unpacked and injected into the local virtual network.

Figure 5 shows an example of two virtual networks. Each virtual network has two virtual hosts (VH) and a ghost host (GH) offering a remote bridge endpoint (RBE), attached to the UNIX-Socket. The Ethernet frames that are extracted from a virtual network are sent across a transport network (cloud) encapsulated into a transport protocol. As transport protocol TCP/IP or UDP/IP is assumed.

## 5 EXAMPLE SETUP

Figure 6 shows an example setup of a DVCSL for two students (Student A and Student B). It is

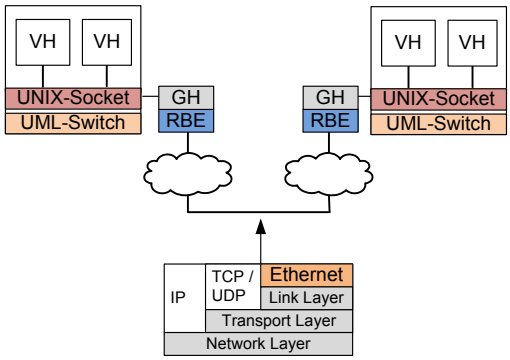


Figure 5: Connecting two virtual networks.

presumed that each student uses his/her own computer connected via a local or wide area network (LAN/WAN). In the example setup, Student A is at public IP address 192.168.2.102 and can reach Student B at IP address 192.168.2.101, and vice versa. Each student locally runs a VCSL, but a student may also run a native Linux operating system, thereby eliminating the need for running the first virtualization layer shown in figure 1. In the example, each student sets up a Netkit environment consisting of two virtual hosts (VH) connected in a local virtual network.

For example, Student A starts VH1 by issuing the following commands:

```
#Start VH1 connected to network netA
vstart VH1 --eth0=netA
#Setup network interface on VH1
ifconfig eth0 192.168.2.200 netmask
255.255.255.0 up
```

In a similar way, Student A starts VH2, and Student B starts VH3 and VH4 connected to netB. The local networks are connected as a distributed local subnet by using the approach presented in section 4. Student A connects to the remote bridge endpoint of Student B, using the user application named plug as follows:

```
#Start remote connection to Student B
plug --source-ip 192.168.2.102 \
--source-port 53838 \
--destination-ip 192.168.2.101 \
--destination-port 32000 \
--uml-switch-socket \
/path/to/vhub_USERNAME_netA.cntc
```

Student B will do the same, connecting to theremote bridge endpoint of Student A. From now on, netA and netB are connected with each other, and all four virtual hosts can reach each other.

In the following, we provide two scenarios that demonstrate the correct and secure operation of the DVCSL shown in figure 6. In these scenarios, we

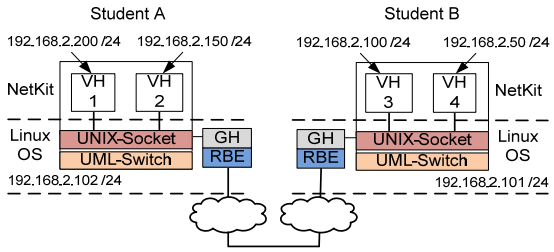


Figure 6: Example setup.

use Ping, a well-known tool for testing the connectivity between two computer systems by sending an echo requests and receiving an echo reply. We use Wireshark (www.wireshark.org), a tool for network protocol analysis, to visualize the recorded network data.

## 5.1 Scenario 1

Scenario 1 demonstrates that virtual host VH1 of student A can communicate with virtual host VH4 of student B. Moreover, scenario 1 shows that the virtual hosts of student A and B act as if they wereconnected in the same network segment (hub-like behavior).

Student A runs Ping on VH1, trying to connect to VH4. The expected result is that VH1 first sends an address resolution request, using ARP (Address Resolution Protocol), to obtain the Media Access Control (MAC) address of VH4. This procedure is common for IP/Ethernet-based networks to obtain the Ethernet address of a host when only its IP address is known. Once the MAC address of VH4 is obtained, VH1 sends an echo request which is answered by an echo reply. The network data which is sent between the Linux operating systems should be encapsulated in a transport protocol.

Figure 7 shows the network data captured at virtual network interface eth0 of VH1. As expected, an ARP request is sent to obtain the MAC address of virtual host VH4 (No. 1). Receiving the ARP reply indicates that both hosts are in the same local subnet (No. 2). Afterwards, the echo request is sent which is replied by VH4 (No. 3-4). The IP addresses of the virtual network interfaces of VH1 and VH4 are correctly shown as source and destination addresses.

Figure 8 shows the network data that is sent between VH1 and VH4, captured at the network interface of the underlying Linux operating system on student A's computer. Due to the encapsulation of the network data in a transport protocol (shown as UDP), the IP addresses of the Linux operating systems are correctly shown as source and destination addresses. The payload of the first UDP packet has a size of 42 bytes which corresponds to the size of the ARP request as seen in figure 7.

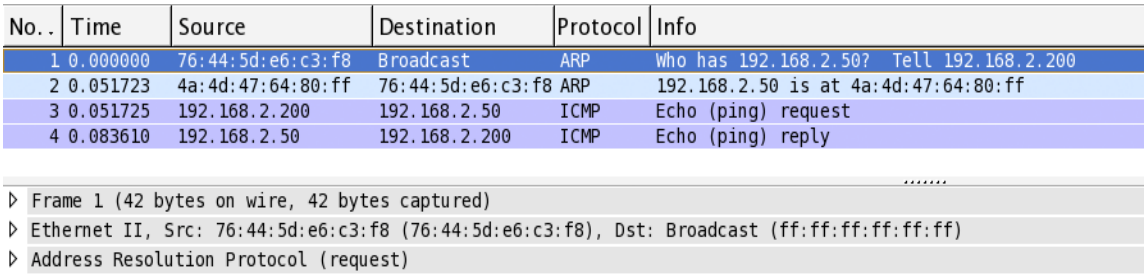


Figure 7: Wireshark capture inside the virtual lab (scenario 1).

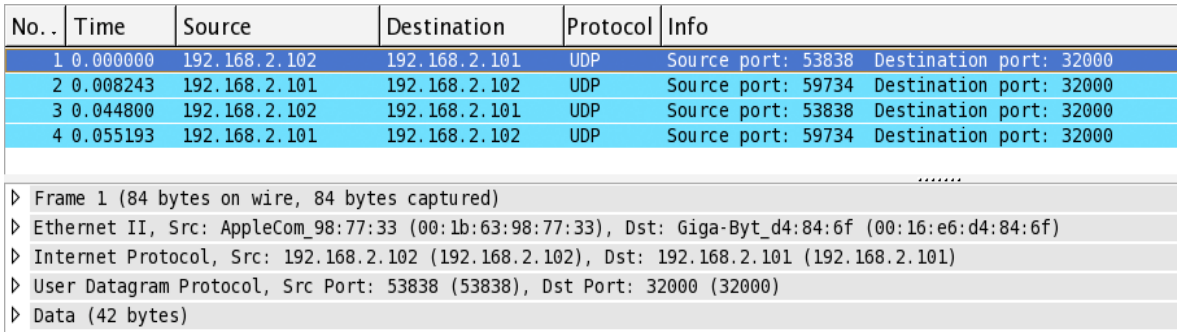


Figure 8: Wireshark capture outside the virtual lab (scenario 1).

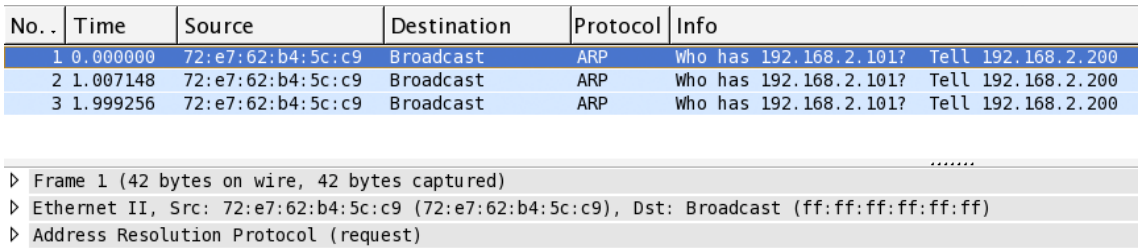


Figure 9: Wireshark capture inside the virtual lab (scenario 2).

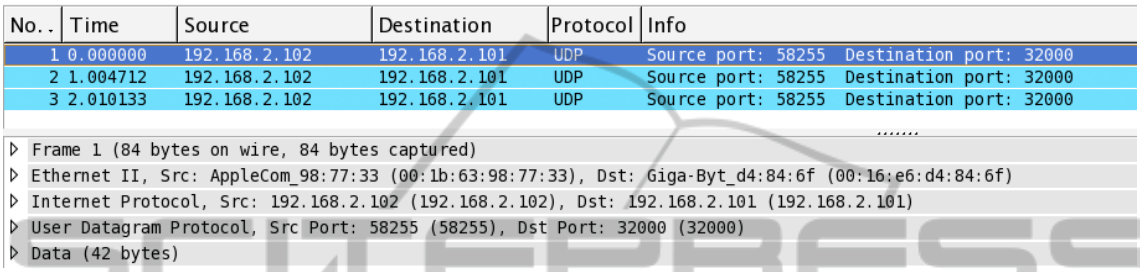


Figure 10: Wireshark capture outside the virtual lab (scenario 2).

We conclude that the virtual hosts of the two students are capable of communicating with each other and that the network data is sent encapsulated in a transport protocol.

## 5.2 Scenario 2

Scenario 2 demonstrates that virtual host VH1 of student A cannot communicate with the Linux operating system of student B, although their IP addresses are located in a common subnet (192.168.2.0/24). The expected result for this scenario is that no communication between VH1 and the Linux operating system of student B is possible due to the strictly separated networks. VH1 sends ARP requests to obtain the MAC address of Student B's Linux operating system, but these ARP requests will never be received by the Linux operating system of Student B. Instead, they are encapsulated and sent to the remote bridge endpoint of Student B.

Figure 9 shows that VH1 tries to obtain the MAC address of the Linux operating system of Student B by sending ARP requests several times. These requests are not answered due to non existing connectivity to the network of the Linux operating systems.

Figure 10 shows that the Linux operating system receives the ARP request encapsulated in the transport protocol. The payloads of the UDP packets have a size of 42 bytes which corresponds to the size of the ARP requests as seen in figure 9. Due to the encapsulation, the Linux operating system does not recognize the packets as ARP requests. It therefore does not send a reply, but sends the network data to the remote bridge endpoint. The virtual network of student B however does not contain a VH with IP address 192.168.2.101, and hence no echo reply will be sent. (This also shows that student B may add a VH with this IP address. Hence there is no restriction on the IP addresses that can be used for VH's inside the DVCSL.)

We conclude that no connectivity is possible between the Linux operating systems and the virtual hosts, although they use the same subnet. This shows that the DVCSL is securely isolated from the outside world.

# 6 FUTURE WORK

In this paper we showed that two isolated, distant, virtual networks can be connected in a DVCSL transparently and securely. For connecting to a remote bridge endpoint, a student needs to know the IP address and port of the computer where the remote bridge endpoint is located. A student may also need to do additional network configuration for allowing communication between the remote bridge endpoints. This can for example be necessary when network address translation (NAT) is used, which then requires port forwarding.

With the point-to-point connection of two virtual networks, also more than two virtual networks can be connected. However, this requires some self organization by the students to avoid the appearance of circular flow of network data.

Our current research addresses these issues by adding a central authority that connects and manages multiple VCSL's and forwards data between them.

We showed that traffic inside the DVCSL is isolated from the outside world. However, a malicious user outside the DVCSL can monitor the DVCSL traffic that is sent over the internet. Such a malicious user can access the encapsulated OSI-layer 2 data by using the techniques as described in this paper, and distribute the data outside the DVCSL. Access to our OSI-layer 2 traffic is possible only by intentional installation and configuration of additional software components which implement a remote bridge endpoint. Typical TCP/IP stack configurations do not contain a remote bridge endpoint. Despite of this we suggest to deploy existing encryption libraries like SSL for future implementations of our DVCSL in order to make our network traffic completely inaccessible for non-DVCSL systems.

## 7 CONCLUSIONS

We presented a DVCSL in which remote students can perform network security exercises inside an encapsulated common networking environment. The DVCSL is built by connecting distinct VCSL's transparently at OSI-layer 2 across an arbitrary TCP/IP-based WAN infrastructure like the internet. To implement this connection, we designed a software component called ghost host with an interface to access local virtual network traffic. The ghost host can extract and inject Ethernet frames. We used the concept of a remote bridge endpoint to transport all local OSI-layer 2 traffic between remote ghost hosts across a TCP/IP-based WAN. As a proof of concept, we demonstrated an example setup which shows that both major goals of our effort are reached: the remote virtual networks are connected transparently at OSI-layer 2 and no intentional or unintentional damage can affect systems not participating in the DVCSL.

Summarized our DVCSL will allow remote students to attend practical courses in network security similar to courses performed in a real safeguarded networking laboratory on a technical level. As an overall result, this is a considerable step towards combining the advantages of distance education and on-site training.

## REFERENCES


Bishop, M. and Heberlein, L. T. (1996). An Isolated Network for Research. *19th National Information Systems Security Conference*, 22-25.

Border, C. (2007). The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes. *ACM SIGCSE Bull.*, 39(1), 576-580.

Bullers, W. I., Burd, S. and Seazzu, A. F. (2006). Virtual machines - an idea whose time has returned: application to network, security, and database courses. *Proc. SIGCSE Techn. Symp. on Computer Science Education*, 102-106.

Comer, D. E. (2001). *Computer Networks and Internets, with Internet Application* (3rd ed.). New York, Prentice Hall.

Damiani, E., Frati, F. and Rebeccani D. (2006). The Open Source Virtual Lab: a Case Study. *Proc. Workshop on Free and Open Source Learning Environments and Tools*, 5-12.

Dike, J. (2006). *User Mode Linux*. New Jersey, Prentice Hall.

Drigas, A. S., Vrettaros, J., Koukianakis, L. G., and Glentzes, J. G. (2005). A Virtual Lab and e-learning system for renewable energy sources. *Proc. WSEAS Int. Conf. on Educational Technologies*, 149-153.

Gaspar, A., Langevin, S. and Armitage, W. D. (2007). Virtualization technologies in the undergraduate IT curriculum. *IEEE IT Professional*, 9(4), 10-17.

Hay, B. and Nance, K. L. (2006). Evolution of the ASSERT computer security lab. *Proc. Coll. for Information Systems Security Education*, 150-156.

Hu, J., Cordel, D. and Meinel, C. (2005). Virtual machine management for Tele-Lab "IT-Security" server. *Proc. IEEE Symp. on Computers and Communications*. 448-453.

Jakab, F., Janitor, J. and Nagy, M. (2009). Virtual Lab in a Distributed International Environment – SVC EDINET. *Proc. Int. Conf. on Networking and Services*, 576-580.

Keller, J. and Naues, R. (2006). Design of a virtual computer security lab. *Proc. IASTED Int. Conf. on Communication, Network, and Information Security*, 211-215.

Krishna, K., Sun, W., Rana, P., Li, T. and Sekar, R. (2005). V-NetLab: a cost-effective platform to support course projects in computer security. *Proc. Annual Coll. for Information Systems Security Education*, 1-7.

Lahoud, H. A. and Tang, X. (2006). Information security labs in IDS/IPS for distance education. *Proc. Conf. on Information Technology Education*, 47-52.

Li, P. (2009). Exploring virtual environments in a decentralized lab. *ACM SIGITE Research in IT*, 6(1), 4-10.

Mattord, H. J. and Whitman, M. E. (2004). Planning, building and operating the information security and assurance laboratory. *Proc. Annual Conf. on Information Security Curriculum Development*, 8-14.

O'Leary, M. (2006). A laboratory based capstone course in computer security for undergraduates. *Proc. SIGCSE Techn. Symp. on Computer Science Education*, 2-6.

Pizzonia, M. and Rimondini, M. (2008). Netkit: easy emulation of complex networks on inexpensive hardware. *Proc. Int. Conf. on Testbeds and Research Infrastructures for the Development of Networks & Communities*, 1-10.

Rimondini, M. (2007). *Interdomain Routing Policies in the Internet: Inference and Analysis.* PhD thesis. Rome, Roma Tre University.

Schreiner, R. (2009). *Computer-Netzwerke.* Munich, Hanser Verlag.

Schürmann, B. (2004). *Grundlagen der Rechner-kommunikation*, Wiesbaden, Friedr. Vieweg & Sohn Verlag.

Stevens, W. R., Fenner, B. and Rudoff, A. M. (2003). *UNIX Network Programming Volume 1: The Sockets Networking* (3rd ed.). Boston, Addision-Wesley.

Vranken, H. and Koppelman, H. (2009). A virtual computer security lab for distance education. *Proc. IASTED Int. Conf. on Internet and Multimedia Systems and Applications*, 21-27.

Yang, T. A., Kwok-Bun, Y., Liaw, M., Collins, G., Venkatraman, J. T., Achar, S. and Sadasivam, K. (2004). Design of a distributed computer security lab. *J. of Computing Sciences in College*, 20(1), 332-346.