V-NetLab: An Approach for Realizing Logically Isolated Networks for Security Experiments[*]

Weiqing Sun *Varun Katta* *Kumar Krishna* *R. Sekar*
Department of Computer Science
Stony Brook University, Stony Brook, NY 11794

Abstract

Cyber security experiments with potentially malicious software can possibly damage the testbed environment and "escape" into the Internet. Due to this security concern, networks used in such experiments are often totally isolated from production networks and the Internet. This choice, however, precludes remote access to testbeds used for security experiments, thus requiring costly duplication of equipment, manpower and expertise at sites that experiment with malicious software. We propose an alternative approach that is aimed at providing a degree of safety comparable to that of physically isolated testbeds while still permitting remote connectivity. Our approach relies on logical isolation of networks used in different security experiments using network virtualization at the datalink layer. We have implemented this approach into a platform (V-NetLab), and the responses from testbed users have been very positive.

1 Introduction

Teaching and research in practical cyber security requires experimentation with potentially dangerous software on contemporary networks. Experimentation with live malware is particularly challenging since there is a possibility that malware may "escape" the confines of a laboratory and spread over the Internet. Although platforms such as the PlanetLab [3] and Emulab [2] exist today to support experimentation with large-scale network services, security-related experiments are either discouraged or altogether disallowed, especially if they involve intentional malicious behavior.

Even if malware is deployed on a tightly controlled network, and prevented from accessing the Internet by a firewall, there is no guarantee that malware will not access (or escape into) the Internet. In particular, malware may discover and exploit vulnerabilities in the very same network components that are supposed to prevent it from accessing the Internet, such as VLAN switches, routers and firewalls. Alternatively, it may exploit vulnerabilities in network monitoring and intrusion detection applications such as Snort, Wireshark (formerly Ethereal), tcpdump or other network-exposed applications that may be run on the network used for security experiments. Numerous CVE vulnerabilities have been reported involving these applications' components in the past several years, and hence the likelihood of finding exploitable vulnerabilities is non-negligible. When combined with the large magnitude of damage that can result from malware escaping to the Internet, the overall risk may be unacceptable. These risks grow in proportion to the size of experimental testbeds — for instance, an oft-voiced concern about security experiments in the context of the proposed GENI infrastructure is that it should not become the world's most powerful botnet!

A typical approach for mitigating the security concerns mentioned above is to physically disconnect experimental networks from the Internet, but this requires researchers to have physical access to these networks — something that is not possible in the context of most large-scale testbeds. To mitigate this difficulty, DETER [1] (which is based on Emulab) can allow remote access to hosts used in security experiments, while mitigating risks by deploying firewalls, network access control mechanisms and/or IDS/IPS. However, errors in configuring these mechanisms may result in the possibility that malicious code may escape into the Internet. As a result, DETER recommends no remote access for the most dangerous experiments. But disabling remote access for a particular experiment is insufficient, it is possible that malicious code exploit vulnerabilities in the software/hardware components of the testbed infrastructure and escape to other experiments that have remote access.

To provide controlled Internet-access for carrying out

[*]This research was supported in part by NSF grants DUE-0313858, CCR-0208877 and CNS-0627687.

these experiments, we propose a new approach that uses datalink layer virtualization in conjunction with virtualization of hosts involved in security experiments. It is implemented by rewriting every network packet created within a security experiment in such a way as to isolate these packets from the underlying network infrastructure. As a result, malicious packets cannot "escape" outside the experimentation network, and therefore have no opportunities to exploit vulnerabilities in servers, firewalls or routers within production environments. Remote access can be provided by creating a communication tunnel from an experimenter's workstation to the experimentation network.

The key insight behind our approach is that *network virtualization* can be used to provide security in much the same way as *host virtualization.* Just as a VMM mediates all accesses made by a guest OS to host hardware, our approach relies on a packet rewriter deployed on the host machine that mediates all accesses to the physical network made by a guest VM. We point out that while today's security technologies such as anti-virus and application firewalls can provide some defense against malicious code, they are not considered strong enough to run malware experiments. This is because malware can potentially compromise its operating system, and subsequently disable or defeat all defenses deployed on this OS. In contrast, since a VMM is a much simpler piece of software that provides a much narrower interface to a guest OS as compared to the complexity of an application-to-OS interface, it is believed to provide an adequate level of security for experiments involving malware. In the same manner, our packet rewriter is a small piece of code with very simple functionality, and hence can be relied upon to ensure isolation of the virtual networks from the underlying physical network.

Our packet rewriter encapsulates the guest VM's network packets in a manner that they will no longer be interpreted by the underlying network fabric. In particular, consider a datalink layer packet p from a host A to another host B, where A and B are part of a security experiment. In V-NetLab, both A and B will be implemented as guests on host machines A_h and B_h respectively. The packet p is intercepted by a packet rewriter (implemented using a kernel module) on host A_h, which generates a new datalink layer packet p' with the source address of A_h, destination address B_h, and a protocol identifier ETH_P_VNETLAB that is unused in the (physical) testbed. The payload of p' is the entire packet p. On B_h, the kernel hands packets with the protocol identifier of ETH_P_VNETLAB to our packet rewriter, which inverts the above transformation and hands p to the guest B. Due to the fact that p' looks like any other datalink layer packet from A_h to B_h, it is highly unlikely to compromise any components on the physical testbed that operate at these datalink layer. Moreover, since the protocol identifier of ETH_P_VNETLAB is unknown to these components, they are unlikely to inspect or process its payload. As a result, components on the physical network are highly unlikely to be compromised (or affected in any way) by network traffic generated as part of security experiments. Additionally, the entire payload of p' can be encrypted in order to ensure that its contents remain confidential, or to ensure that the resulting payload looks essentially random (i.e., uncorrelated with the original packet contents) and hence cannot predictably be used to exploit vulnerabilities on these devices/services.

A benefit of virtualization at the datalink layer is that it permits the use of any layer-3 protocol within the security experiments, including IP, ICMP, ARP, etc. Moreover, since the packets transmitted by guest OSes remain encapsulated on the physical network, it is possible for different virtual networks to use overlapping IP addresses without interfering with each other. Indeed, the management of IP addresses on the virtual network is totally under the control of the user of the virtual network; these addresses need not be allocated or approved by the operators of the underlying physical network. We remark that our mechanisms for remote connectivity operate independent of such overlaps in IP address space.

In the rest of this paper, we first describe the design and implementation of our approach in Section 2. Several issues about our approach are discussed in Section 3. Related work is discussed in Section 4, followed by the concluding remarks in Section 5.

2 Design and Implementation

2.1 V-NetLab System Architecture

We use Linux as the host OS to simplify development of virtual network control and management software. The virtual machines (guest OSes) themselves may run Linux, Windows, or any other OS supported by the virtual machine software. Our implementation uses VMware for virtualization, since it provides seamless support for different guest OSes, strong isolation, and reasonably good performance.

We have implemented V-NetLab framework on a hardware platform consisting of dual-processor workstations connected together by a switched gigabit network. Also included in the platform is an NFS server with a large enough disk to accommodate many virtual machine images. Since our platform is relatively small (currently about 10 workstations), our network infrastructure consists of a gigabit Ethernet switch. With this physical infrastructure, we have been able to support around 50

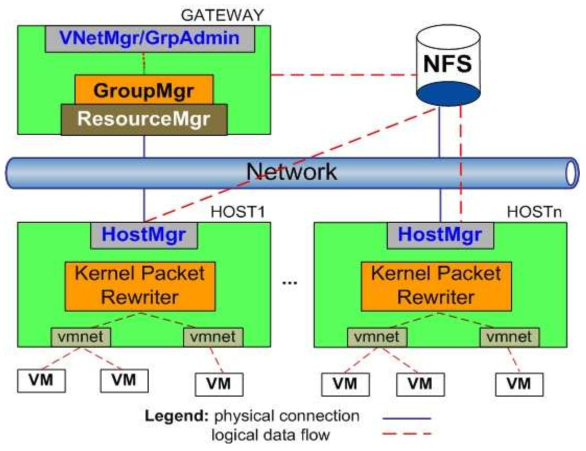

Figure 1: V-NetLab System Architecture

experiments, each consisting of up to ten hosts. These experiments were carried out by students in the context of network and system security courses, and involved a variety of tasks including network mapping and sniffing, network intrusion detection, experiments with malicious code, exploit development, etc. [1]

We note that scaling to a larger testbed is primarily a resource management and mapping problem, i.e., it requires the development of algorithms and techniques to map logical networks onto the physical hosts. Development and implementation of these techniques is ongoing effort. At present, we assume that this mapping has been generated and given to the V-NetLab runtime system, which is responsible for managing the physical infrastructure and interacting with remote users. This runtime system provides management commands that may be used by V-NetLab users to create virtual networks and instantiate them, query their status, etc. The components of V-NetLab runtime system are shown in Figure 1.

- *VNetMgr* is the user interface program for users to register, deregister, start, query, and shutdown their networks.
- *GrpAdmin* is the user interface program for administrators. This interface allows the administrator to add/delete users, register/deregister networks for the users under his domain, create/delete teams, and query the usage statistics of hardware resources for the running networks, etc.
- *HostMgr* is a daemon which runs on each of the physical nodes which host virtual machines for V-NetLab. The HostMgr on a host is responsible for starting up and shutting down virtual machines that reside on that host, as well as configuring the virtual networking components (i.e., packet rewriters) on that host.
- *GrpMgr* is a central daemon which listens to the user interface programs, processes their requests and relays them to HostMgr. Its core component *ResourceMgr* maintains information about all the virtual networks which are currently running, current resource usage, residual computing resources available, etc. It is also responsible for instantiating new virtual networks by mapping their components (hosts, hubs, and switches) onto available resources. It interacts with the HostMgrs to start up the virtual machines that have been mapped to that host.
- *Kernel Packet Rewriter* is a packet handler that encapsulates outgoing packets from virtual machines, and inverts this process for incoming packets. The following section expands on the function and operation of the rewriter and other V-NetLab components for network virtualization.

[1]Details of some security projects are described in [5].

2.2 Datalink Layer Network Virtualization

VMware virtual network interfaces support host-only mode, bridged mode and NAT mode. The bridged mode automatically extends the virtual machine network interface onto the LAN of physical machine, while NAT provides limited connectivity to the underlying physical network. Thus, both these choices are inconsistent with our isolation goal. Hence our approach is based on the host-only mode, which provides connectivity with the host OS but nothing else. The virtual Ethernet adapter inside the guest OS is associated with a virtual network interface `vmnetX` on the host OS, where `X` is a number. Note that multiple guests can share the same host `vmnet` device. In that case, the `vmnet` device acts like a network hub.

A virtual network typically consists of a number of virtual machines distributed over a set of physical hosts. Virtual machines belonging to the same subnet may be distributed over multiple physical hosts in the testbed. In that case, our packet handler is responsible for transparent transmission of packets among virtual machines via physical hosts. In order to achieve this, the packet handlers maintain the mapping between the virtual network topology (also called *logical topology*) and the physical topology. We use two data structures to maintain the mapping information. Given a logical topology, logical interfaces that are directly connected together by one or more hubs form a *sibling closure* (SC). A logical host interface in an SC should be able to listen to network traffic originating at (or destined to) any of the other logical host interfaces in the same SC. The SCs may then be connected together using (logical) switches. Therefore, the logical topology provided by the user can be divided into multiple SCs.

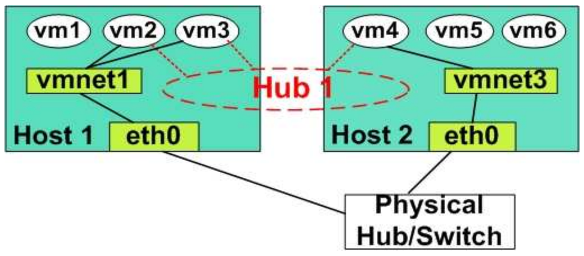

Figure 2: Virtualization at Datalink Layer

Another important data structure is *Participation Table* or *PT*, which lists the set of vmnet interfaces that belong to an SC. This is generated during the start-up phase of the virtual network. By definition, each vmnet interface in an SC is able to listen to conversations involving any other vmnet interface of that SC. The packet rewriter is responsible for ensuring this by relaying packets across the vmnet interfaces of different hosts that are part of the same SC. Figure 2 illustrates these concepts. To reduce clutter, only a single vmnet interface is shown for each host, but in reality, each physical host can support about 100 vmnet interfaces. In this figure, vm2, vm3 and vm4 are supposed to be on the same logical hub. Since vm2 and vm3 have been mapped to the same physical host, they can share a single vmnet interface, called vmnet1 in the figure. vm4 needs a separate vmnet interface on Host 2, called vmnet3. The interfaces vmnet1 and vmnet3 are participants in an SC. Thus, our packet rewriter will ensure that any packet from any of the hosts vm2, vm3 and vm4 would be relayed on both vmnet1 and vmnet3. Virtual network packets observed by the packet rewriter that do not originate from the participants in the SC will be dropped by the rewriter to ensure isolation. An exception to this rule is made for packets originating on the respective physical host. For instance, Host 1 can transmit packets to vm2 or vm3 — the packet rewriter will handle them appropriately. This ability plays an important role in providing external tunnels into virtual networks.

Virtual network packets that need to be forwarded on a remote vmnet interface have to be transmitted on physical wire through host physical interface (e.g., `eth0`). To transparently transmit packets generated by a virtual machine on to the physical wire, we need a mechanism to grab every packet arriving at the vmnet interface and forward it on to the physical host's network interface (eth0) as specified in PT. Moreover, the packet needs to be captured at eth0 interface of the receiving physical host and then forwarded to participating vmnet interfaces on that physical host. These are ensured by our packet rewriter, which is implemented as an extra layer of packet handler (using *dev_add_pack* in Linux) to intercept all Ethernet packets, and then encapsulate the packet using a special data link layer protocol *ETH_P_VNETLAB* with the source/destination MAC addresses as the MAC address of the sending host network interface and receiving host interface[2]. On the receiving host, the packet handler simply removes the encapsulated header and forwards the packet to corresponding vmnet interfaces according to the PT.

2.3 Controlled Remote Access to Virtual Networks

V-NetLab allows tunnels to be created from external networks to a host on a virtual network. Although such tunnels can be created for any purpose, our current implementation targets the most common case, namely, that of creating a tunnel from a virtual network user's workstation (which would typically be outside of the physical testbed infrastructure) to any of the virtual machines on the virtual networks belonging to that user. If multiple users share a virtual network, then each of them can independently (and simultaneously) make use of this capability.

Note that, in order to access V-NetLab, a user is required to first establish an SSH session with a gateway machine to the testbed. Hence our approach relies on using this connection to tunnel network packets to the target virtual network. A natural way to do this is to use the tunneling capability provided by ssh, which allows data targeted at a specified TCP port on the ssh client machine to be forwarded so as to reach a destination port on the server machine and vice-versa. Using this capability, a tunnel is created from port A on the user's workstation to port B on the gateway. Next, another SSH tunnel is created from port B on the gateway to port C on the physical host that hosts the target virtual machine. Finally, Linux `iptables` is used to forward packets from port C to port 22 (i.e., the SSH port) of the target virtual machine. Figure 3 illustrates this process. Our management infrastructure automatically picks suitable values for ports A, B, and C, and sets up the tunnels. It also adds additional iptables rules on the gateway to ensure that a user can only send packets to the ports corresponding to the tunnels created by her. This ensures that one user cannot (accidentally or intentionally) obtain network connectivity with the virtual networks belonging to other users.

[2]The encapsulation operation will increase the length of the original packet, and so there is a possibility that the resulting packet will be dropped by switches or hosts on the physical network due to MTU size limitations. If this happens, the packet rewriter has to provide a fragmentation and reassembly function as well. We did not encounter MTU size problems on our implementation platform and hence have not implemented this fragmentation/reassembly function.

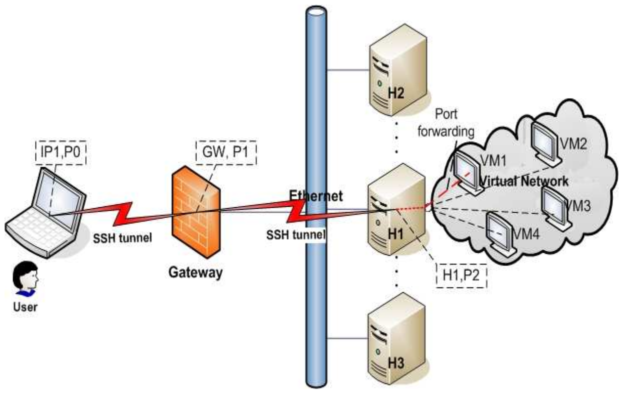

Figure 3: Enabling Remote SSH Access

3 Discussion

3.1 Security Analysis

Virtual networks realized using our approach are isolated from each other, and from physical network infrastructure. A mechanism for controlled external connectivity into this network is provided, but this mechanism cannot be used by a malicious virtual machine to send data to arbitrary external hosts; instead, data may be sent only to a single host and port number to which a tunnel has been explicitly set up.

Another way to think about our network virtualization approach is that it enforces a strong form of "default deny" policy: network packets belonging to a virtual network cannot be seen by any host (virtual or physical) unless explicitly permitted through the deliberate process of network virtualization and tunnel creation.

We point out that security based on isolation provided by virtual machine software can become weakened if malware running inside the virtual machine can exploit vulnerabilities in virtual machine software (e.g., hypervisors) so as to attack the host environment. As far as we know, such attacks have not become a real-world threat. Moreover, since virtual machine software is simpler and provides a narrower interface to a guest OS as compared to the application-to-OS interface, it is believed to be feasible to harden this layer.

Another target for malware attacks is the packet rewriter, which has to operate on packets that may be created by malware. However, due to the small size and conceptual simplicity of this module, we believe that the likelihood of finding exploitable vulnerabilities in this module is rather small.

As a testbed for security experiments, V-NetLab might become an attractive target for attacks. We rely on best security practices to mitigate this threat, such as the installation of a minimal set of applications and services on the gateway machine, providing only a restricted shell for users that permits them to invoke only the commands relevant for managing their virtual networks, etc.

In order to enhance usability, V-NetLab provides a controlled remote access functionality, which can potentially be used to transport malicious code or data outside of the virtual networks, or perhaps to carry out an attack on external hosts. Our mechanisms are designed so that this is possible only with active involvement by the user of a virtual network. They have to explicitly set up these tunnels; and even then, connectivity is provided only to their desktops. If malicious code/data needs to reach elsewhere, the user needs to propagate the data further from his workstation, which requires additional deliberate actions on his behalf. In other words, if the user of a virtual network is malicious, then such outflow of malicious data/code can easily occur. Otherwise, we believe that the risk is very small. For self-spreading worms, they would need propagation channels such as email, IRC and instant messaging. The controlled tunnel created in V-NetLab is specific to a particular application, e.g., ssh. And only the ssh client on the user's desktop is exposed to the virtual network. Therefore, the likelihood of being used as an automatic worm spreading method is greatly minimized.

3.2 Limitations

Currently, V-NetLab does not yet provide advanced resource management capabilities, e.g., reservations. It also does not automate the mapping of large virtual networks onto multiple physical hosts, instead relying on configuration files associated with a virtual network to provide most of these details. Our ongoing work is concerned with on-the-fly decomposition of these networks based on available resources and instantiating them.

Our V-NetLab is currently LAN-centric. This does not mean that it cannot support complex topologies, but it certainly means that our focus has been on relatively small networks. In fact, it can support networks that involve multiple hops between senders and receivers by assigning dedicated virtual machines as virtual routers.

4 Related Work

Although approaches such as [7, 10, 6] have simplified the setup of individual hosts for security experiments, creation of entire networks is still cumbersome. Planetlab [3] is a distributed laboratory that provides convenient management tools to control a large collection of hosts that run identical software. Emulab [2] is another similar approach that provides such facilities. Emulab supports light-weight virtualization, based on FreeBSD Jails. But both approaches do not provide support for running security experiments, where damage can escape to the Internet. Another drawback is that they do not provide the degree of flexibility needed for our approach, where computers running different OSes may need to be hosted on the same physical machine. An alternative mode supported in Emulab is one where physical nodes on the testbed can be dedicated to run a custom OS image. This approach provides the desired degree of flexibility to support security experiments, but does not allow sharing of underlying hardware across multiple OSes.

DETER [1] is a project which aims at building a testbed for security research based on Emulab. In addition to isolating different experiments using VLANs, firewall and intrusion detection system are deployed to prevent potentially malicious experiments from affecting the Internet. However, as mentioned earlier, there exists a possibility that malware may be able to subvert these defenses by exploiting vulnerabilities in network-exposed hardware and software. In contrast, we believe that V-NetLab provides stronger isolation guarantees, while still providing safe external connectivity.

VNET [8] and VIOLIN [4] have some similarity with our work in terms of the support for virtual networks. VNET is concerned with distributed computing applications, and their virtual networks span a wide-area network. Their approach is based on tunneling Ethernet packets over TCP/IP. VIOLIN uses an application-level virtual network architecture built on top of an overlay infrastructure such as Planetlab. They use UDP tunneling in the Internet domain to emulate the physical layer in the VIOLIN domain.

Unlike VNET and VIOLIN, our approach achieves network virtualization at the datalink layer. This provides better performance, as it eliminates the need for higher layers of the protocol from having to process the same packet twice. Moreover, our approach supports multiple virtual networks to share the same set of IP addresses, thus make the management task easier. On the other hand, our approach currently does not support experiments that span across the Internet.

Wroclawski et al [9] propose a collaborative specification of constraints from both experimenters and testbed operators in order to achieve the balance of both security and usability. Our approach does not make any assumption that experimenters provide constraint requirement of their experiments, and can provide satisfactory security and usability even in face of malware experimentation. On the other hand, our approach can potentially achieve better usability (e.g., controlled tunnel creation can be made less constrained.) with accurate constraint information from the experimenters.

Krishna et al [5] presented our earlier version of V-NetLab with the focus on using it to support security course projects in a class setting. This paper focuses on our approach for realizing logically isolated networks for supporting security experiments.

5 Conclusion

In this paper, we presented our approach for realizing virtual networks for security experiments. The datalink layer virtualization approach provides strong isolation for virtual networks, while providing the flexibility for establishing connectivity to external networks in a controlled way.

We implemented this approach into V-NetLab framework and have so far used it in several security courses in our department to provide hands-on projects for students. Student response to date has been very positive.

6 Acknowledgments

We would like to thank our shepherd Jelena Mirkovic and the anonymous reviewers for their insightful comments.

References

[1] Deter. http://www.isi.edu/deter.

[2] Emulab. http://www.emulab.net/.

[3] Planetlab. http://www.planet-lab.org/.

[4] Xuxian Jiang and Dongyan Xu. Violin: Virtual internetworking on overlay infrastructure. In *International Symposium on Parallel and Distributed Processing and Applications (ISPA) 2004*, 2004.

[5] Kumar Krishna, Weiqing Sun, Pratik Rana, Tianning Li, and R. Sekar. V-netlab: A cost-effective platform to support course projects in computer security. In *Proceedings of 9th Colloquium for Information Systems Security Education*, June 2005.

[6] Scott D. Lathrop, Gregory J. Conti, Daniel J. Ragsdale, and Wayne Schepens. Information warfare in the trenches: Experiences from the firing range. In *WISE3: 3rd World Conference on Information Security Education*, 2002.

[7] Jean Mayo and Phil Kearns. A secure unrestricted advanced systems laboratory. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 165–169. ACM Press, 1999.

[8] Ananth I. Sundaraj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *USENIX-VM '04: 3rd Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[9] John Wroclawski, Jelena Mirkovic, Ted Faber, and Stephen Schwab. A two-constraint approach to risky cy-

bersecurity experiment management. Invited paper at the Sarnoff Symposium, April 2008.

[10] Alec Yasinsac, Jennifer Frazier, and Marion Bogdanov. Developing an academic security laboratory. In *NCISSE '02: 6th National Colloquium for Information System Security Education*, 2002.