2009

Virtualized network framework solution to collecting private research data NEMESIS: Network Experimentation and Monitoring in Environments Safely In-Situ

Alexander Nicholas Pease
Iowa State University

Virtualized network framework solution to collecting private research data NEMESIS:

Network Experimentation and Monitoring in Environments Safely In-Situ

by

Alexander N Pease

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering & Information Assurance

Program of Study Committee:
Thomas E. Daniels, Major Professor
Doug Jacobson
Cliff Bergman

Iowa State University

Ames, Iowa

2009

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The cyber security research realm is plagued with the problem of collecting and using trace data from sources. Methods of anonymizing public data sets have been proven to leak large amounts of private network data. Yet access to private and public trace data is needed, this is the problem that NEMESIS seeks to solve.

NEMESIS is a virtual network system level solution to the problem where instead of bringing the data to the experiments one brings the experiments to the data. NEMESIS provides security and isolation that other approaches have not; allowing for filtering and anonymization of trace data as needed.

The solution came about from a desire and need to have a system level solution that leveraged and allowed for the usages of the best current technologies, while remaining highly extendible to future needs.

CHAPTER 1. INTRODUCTION

The NEMESIS idea came about from the research that is being conducted at Iowa State University and the needs of other researchers. The computer security and networking fields currently have a problem that is persisting. The problem is that of using research data sets. There are not a lot of public data sets and those that are in existence are designed for a particular problem. If researchers are looking to test a new anomaly based intrusion detection system, the researcher would need a current data set and multiple data sets over a period of time from an enterprise network. Problems persist trying to get this type of trace data from enterprise networks.

Most enterprise networks don't want information about their internal network to make it to the outside of their network. That information is private and they want to protect it, in fact most have security policies preventing that information from leaving the network. In this case the organization might allow for a private trace to be collected on a limited segment of their network, and then allow the researcher access to that trace and the researcher's results could be public but the trace would be private and not allowed to be shared or examined. Another common option is public sanitized data sets, this where an organization has released data to the research community but has sanitized and anonymized the data. Both choices have their own problems.

Private data sets can be hard to negotiate for, and then the results, since the data set is private cannot be verified by another researcher. That is to say that another researcher cannot run his tests on the same data set to verify that what is identified is in the trace. Sanitization is not a complete solution and allows for many different kinds of attacks, and data leakage.

This paper will explain some of the attacks on anonymized network data, and how allowing the release of anonymized network data can leak sensitive private information about the network or individual users.

1.1. NEMESIS

NEMESIS, Network Experimentation and Monitoring in Environments Safely In-Situ, solution is a systems level approach to the problem of using and collecting network trace data that allows for in place experiments to be run. This solution takes the experiments to the trace data owner; instead bring the trace data to the experiment, allowing for the trace data owner to have more control over what information gets release and give some added protection against future attacks on the released network traces since no network traces are released. In figure 1 below a big picture view of NEMESIS is presented. NEMESIS consists of multiple parts. There is a Virtual Framework peace and a policy management peace that is used to control and implement policies on the virtual framework system. This paper address the virtual framework system and makes suggestions at some of the tools for implementing some of the policies. This paper does not discuss in detail or address how the policy management system should be implemented. For the policy management in the paper the network monitoring policies are implemented by the use of software firewalls since standard networking is used to control data flow between host and virtual machine. The policy management will need to be implemented with the given data flow and control piece. This paper proposes one possible implementation of the NEMESIS idea with virtual machines and the use of existing virtual network technologies.

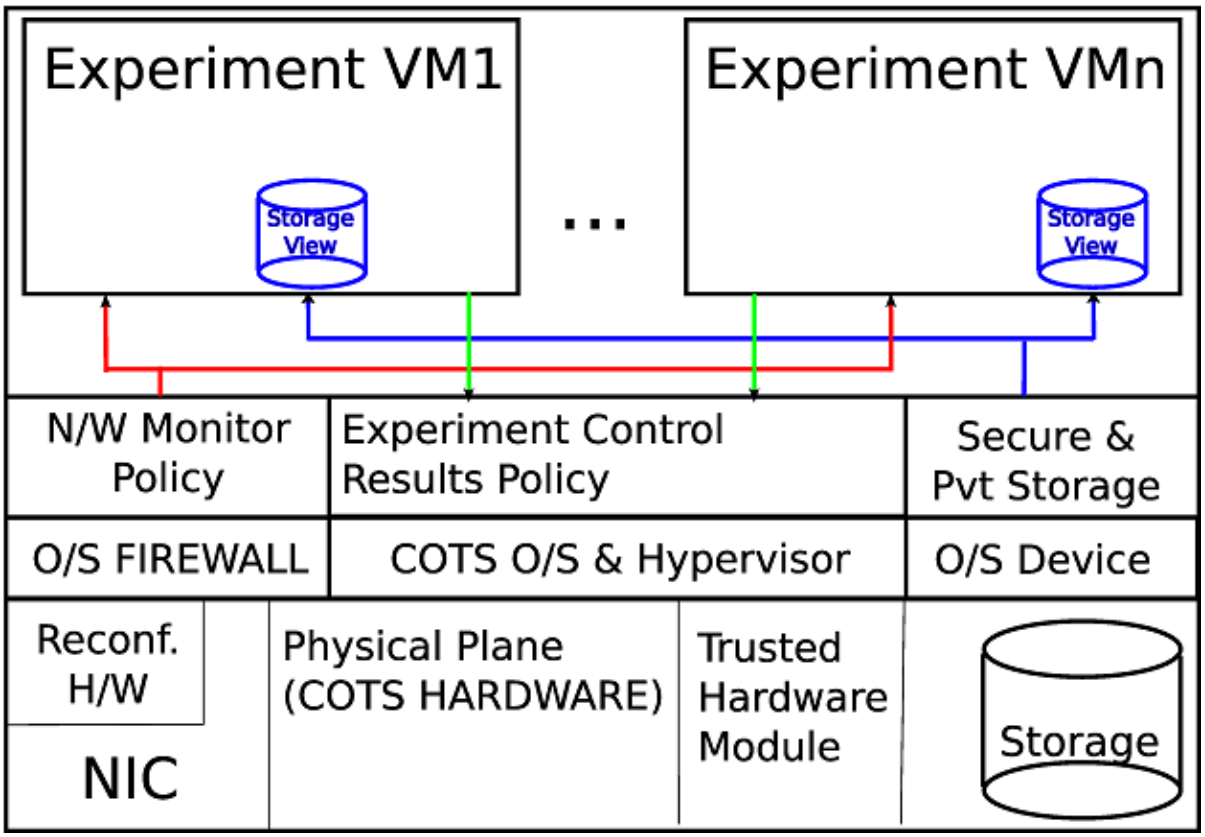

Figure 1. NEMESIS node

A quick overview of the process and to aid in explaining the NEMESIS idea is figure 2 is a conceptual drawing of process flow. The three columns represent the 3 actors that we wish to collaborate during the experiment. The researcher has an idea and then a design is worked out with a developer. The researcher and the trace owner negotiate privacy concerns reach an agreement, and then development beings. Development and local testing begin since once the experiment is deployed the trace owner the researcher will only get the results back. The virtual machine that was developed is deployed to the trace owners NEMESIS node and the experiment begins, and runs autonomously. When the experiment ends, the result are analyzed and if they meet the agreed upon terms, they are released to the researcher for publication.

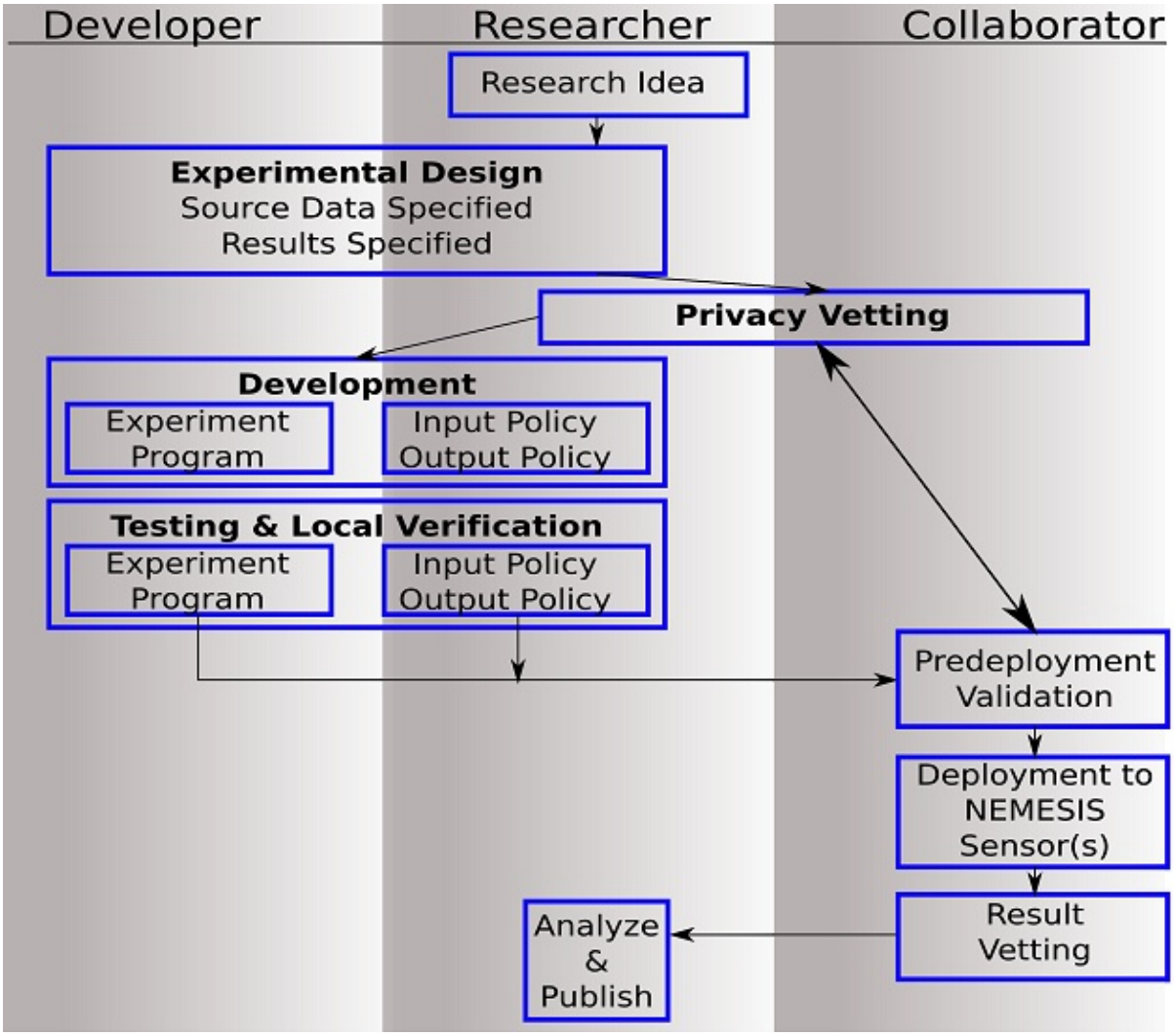

Figure 2. Collaboration example (by Thomas E. Daniels PHD)

NEMESIS utilizes virtual machines to leverage the existing technologies and allows for multiple experiments to be run on the same hardware, reducing the server footprint on an enterprise network and allowing researchers to build their experiment with the tools they want to use and not through a custom programming or query language. This would allow form multiple tools to be run like snort (a common Intrusion Detection System) and custom tools and correlate the results. For the data that is sent back to the researcher.

CHAPTER 2. BACKGROUND

There are many solutions to the problem of dealing with trace data. The solutions range from anonymization to custom languages. This section will address the previous solution, their failures, and the need for a complete solution. This section will discuss the core components of NEMESIS.

Anonymization and sanitization of trace data, was the first solution purposed and is still seen as the front running solution. Anonymization allows for the protection of private network information. Packet body's can be sanitized or removed, IP address can be mapped to new IP address. There are three main ways to do anonymization Partial, Full, and N-Flow anonymization. Recent work has proven that removing payloads and host IP address does not alone protect the privacy of the trace. [Brekne 14]

Partial or pseudo anonymization involves some preservation, where as subnets would be preserved, however changed. So the octets would be the same for any address coming from that octet. For example 123.0.0.0 would map to 444.X.X.X, and 123.1.0.0 would map to 444.340.X.X. In some cases such as Crypto-PAn cryptographic methods are used to anonymize the trace data. These methods are subject to cryptographic attacks on the algorithm. [Brekne 14] Full anonymization is when the IP address fields are randomized and the payload is stripped. When this is done the data set becomes less useful to researchers. [Mirkovic 13]

TCPdpriv is a tool that executes on tcpdump trace files and performs anonymization on these files. TCPdpriv removes sensitive information by operating only packet headers the payload is fully removed. TCPdpriv can do full stripping, prefix-preserving psedo-

anonymization of network trace preserving topology for the researcher incase this is needed for the results. While running TCPdpriv maintains a list of IP address mappings in memory.

Crypto-PAn is a tool much like TCPdpriv allowing for one-to-one maps between IP address in source and result trace and prefix preserving. Because Crypto-PAn is based on cryptographic methods using a key to determine the mapping, as long as the same key is used the same mapping can persist across multiple sessions. The tool is based on Rijndael cipher for cryptography. [Fan 9] Crypto-Pan only works on IP address s and the 8 most common fields of NetFlows, since releasing NetFlows is better than releasing full traces.

Anonymization is a balance between trace owner privacy and effectiveness of the trace to researchers. This is a problem since there are a number of attacks on anonymized traces. The attacks are passive and active. Passive attacks are those that take the public trace data and other public data to infer private data. Active attacks are those that involve doing something while the trace is being collected that can be identified and used to break the privacy of the trace. [Mirkovic 13]

A passive attack example would be using packet length field to identify what websites a particular host has visited. This can be done when all replays are observed within one tcp connection or a summarized Net Flow. The use of ARP data, subnet clustering and publicly available DNS records can be used to get network topologies, determine observation points, and some host matching [Coull 1].

Active attacks include injecting data in at the time of capture that can be pulled out in the public trace and used to break the anonymization. To do this one would spoof source and destination address and then make the header or traffic flow pattern identifiable in the trace

so that the information can be pulled out later. Details are given in [Brekne 14] on who to do injections and active attacks.

The next solution presented to the problem of using private trace data, while preserving the data. This solution relates in many, but limited ways to the solution that this paper purposes. In 2006 at SIGCOMM'06 workshop, SC2D was presented. SC2D is a framework and programming language designed and proposed as a way of bringing research to the data. Instead of getting the data trace from an organization and running tests on that trace. SC2D purposed that the data remain at the organization and that they have a server, which the experiments are run on. This approach utilizes a modular interpretive language. The researcher would develop a module that then would be run and the results returned to the researcher. The framework handles anonymization at a lower abstraction level than what the user programs in. The paper also outlines process for code review, although never tried. The prototype was based and written in BRO IDS, and had performance issues and management of multiple project issues. Conclusion: this a tool in which researchers write their tests in the framework and interpretive language and get the results sent back to them. However if their result rely on correlation of data from existing tools and their tool, the researcher would need something more to be able to get the results they are looking for. {Mogul 5]

The next paper and work that this paper addresses is using secure quarries to query trace data preserving privacy. Although Mirkovic was not the first to suggest secure queries, she presented the idea of dealing with the privacy concerns in the query language and database. The system works by imputing the trace data into a database, then the researchers write their programs to query the data or run off the returned data from the queries. The query language restricts queries on some data fields and some contexts. Results returned

from this purposed solution are not raw packets but aggregate data. The advantage of this system over anonymized trace data lays in fine grained controls and control over the portal to access data. The Portal controls can monitor usage patterns and control if a user is using multiple queries to get at information that when correlated would reveal privacy-sensitive information. This solution has not been prototyped.

No single prevented solution is enough and is a perfect solution. My solution is not perfect either; however it is a complete solution with the ability to be expanded for future needs of researchers.

Virtualization and virtual machines has been seen as a way to run multiple servers on the physical hardware of one server, since space is a comity in a server farm, and server generally don't use all their resources. Virtualization also can add another level of security features, first the servers can have images made of them at a given state and restored if a problem occurs. Virtualization also adds a layer of protection since there is either a hypervisor or host OS (DOM 0). So in order to make system calls all commands go through the hypervisor. All normal security percussion must still be kept and the physical device, hypervisor or DOM 0 must still be hardened. The solution proposed in this thesis can be implanted with either a DOM 0 or a hypervisor approach. Though the implantations very some. With virtual machines the physical device still can control the network adapter and thus the traffic seen or allowed out of any VM can be controlled allowing for firewall in front of every Virtual machine that is not located on the virtual machine. Thus each server can have custom rules in place on the DOM 0, while the DOM 0, could see all traffic going to any VM. Virtualization can be used to help with the security in depth model. In the cpirse of this project's research KVM a DOM 0 approach, and VMWare ESXi a hypervisor

approach were examined. The main difference between a DOM 0 and a hypervisor approach is how resources are managed and allocated. The figure 2 shows how a virtual machine manager works. The virtual machine manager passes and controls the access to the physical hardware.

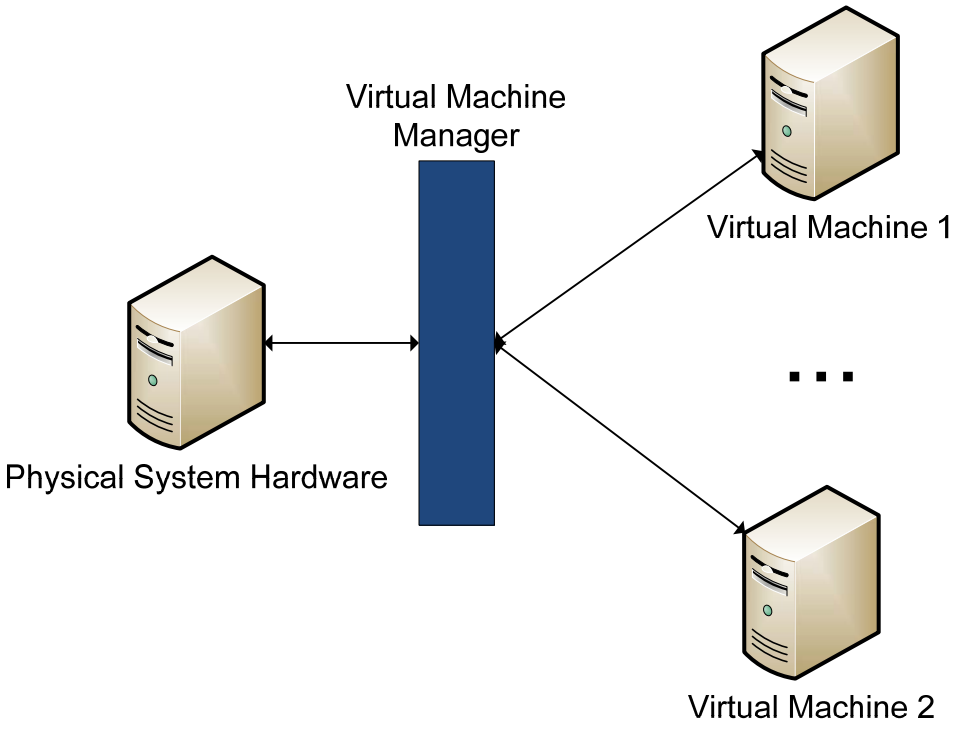

Figure 3. How a virtual machine manager works

CHAPTER 3. IMPLAMANATION AND DESIGN

3.1 Introduction

This chapter focuses on the design and implementation of NEMESIS. It looks at the design goals and constraints of existing technologies, how some technologies presented unique problems, how those challenges were, overcome or a proposed solution that will be future explained in future work. Also a process flow is suggested on how to use the proposed solution with a breakdown of the steps in the process flow.

3.2 Core design components

The core pieces to the design approach are a Virtual Network infrastructure, communication interface, Policy management and enforcement.

3.2.1 Virtual network infrastructure

Hypervisor, a term that finds it roots in mainframes is software that allows multiple host Operating System(OS) to run on the same physical hardware also referred to as virtual machine monitor (VMM). Two types of VMM exist the first type is that which runs directly on the hardware the second runs in a host OS often referred to as DOM 0 or host OS with Virtual machines being referred to as guest OS. As mentioned in the previous section both types could be used to complete this project. The design constraints and needs of the VMM for this project were simple, The VMM needs to be able to run multiple VM experiments at the same time, be highly extendible to future needs and be useable, capable of running multiple different OS from Linux to UNIX to Windows. The VMM must be able to separate

the privileges of one VM from another. There needs to be a way of pulling results off of the VMs from the host OS or have results pushed from VM to Host OS.

Hypervisor type 2 was selected since it allows for better use of current existing technologies running on the host OS and allowed for us of open source technologies that could be expanded or customized in the future for implementations. Running a VMM on a host OS allows for tools to be written in kernel or user space to help with the management or expansion of the solution.

The VMM that was select was Kernel-Based Virtual Machine KVM. KVM is a derivative of KQEMU which is QEMU accelerator that provides a way to run user mode code on the host CPU and some Kernel code on the Host CPU rather than the emulated CPU. KVM utilizes the Intel and AMD CPU virtualization support designed into the micro processor to do some optimization, and it does allow for privately virtualized NICS, hard drives.

One of the most important factors in choosing a type 2 VMM is the ability to pull results off the VM once the VM has been stopped running. With KVM this can be done in multiple ways the first way which is what we outline is to have the VMs create a folder in the root directory of the VM disk image called results. The VM writes its results to this folder, when the VM experiment is completed and the VM is shutdown then the Host OS can mount the disk image as a directory and copy off the results folder to its own results patrician, where the results can be analyzed, and thus allowing for a fully autonomous process in the future. The second possibility is for the Host OS to create a disk image, that is limited in size and then the guest OS mounts this virtual disk and write results to this patrician which are later copied off as describe in possibility one. This paper recommends solution 2 since it utilizes

solution one with adding overall size constraints on the results set. Which is determined to be a problem can be addressed by using a policy, which limits results to the remaining space of the guest OS virtual machine as agreed upon in policy discussions of the experiment. Figure 3 below shows a disk level view of the solution and how the results could be written.

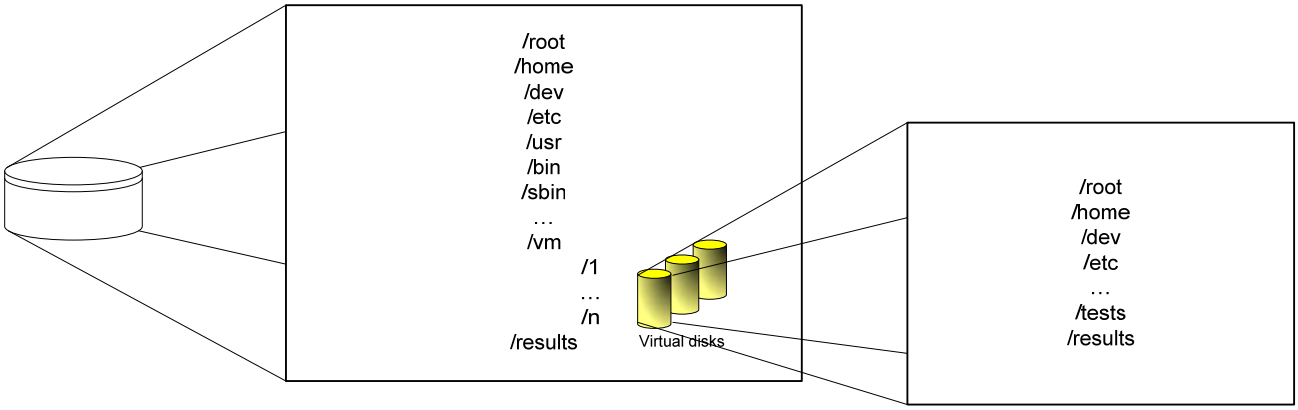

Figure 4. Disk level view of virtual machine infrastructure

3.2.2 Communication infrastructure

Once KVM was selected as the VMM the next step was to determine how the trace data or network traffic would get to the VMs. The obvious choice would be to use virtual network interfaces, for a prototype this is a possibility, however some fine grained filtering and packet alteration, might need something more in the future. The design issues for the communication interface stems from the idea of having multiple VM running on the same physical hardware. Each virtual machine needs to be able to receive different traffic or different portions of traffic. Adding a custom kernel device might be a better choice down the road than using the standard virtual network interface. KVM uses tun/tap devices which

are virtual Ethernet devices; a tun device only functions at the IP layer and not the full Ethernet layer where as a tap device functions at the full Ethernet layer. When creating the virtual interface for the VMs one would need to create tap devices since the full Ethernet headers are needed. In testing it was show that if a tun device is used even if a bridge is in promisc mode, so thus acting as a hub, since a tun device is a layer 3 IP device it functions as a switch only accepting traffic bound for the other side of the tun interface. Interestingly enough the tool used to make static tun/tap interfaces, tunctl, by default is set to create tap interfaces. However, the Debian package that contains tunctl is UML-tools and it is compiled to create tun devices only. On a Debian machine one must compile tunctl from source, on a Fedora machine there is a RPM for tunctl that works fine.

Once you understand how to create and make virtual interfaces, and bind them to a VM it is important to understand the other network tools that will help build a virtual network and firewall the interface so that the experimental VM do not send any traffic out in the instances that they may be bridged to a real network interface for a direct network tap.

There are two different potential types of trace data that could need to reach the VMs. The first potential would be live trace data. The virtual interface is bridge directly to a real interface, or bridged to a virtual interface that is bridged to a real interface. In this Case the Trace data would need to within reason and reliable with expectable packet lose get to the Guest OS's virtual interface. The tool in Linux that is used to make static bridges is "brctrl." Technical details on how bridge control will work are given in appendix A. The basic tools that can be used in Linux to filter on a bride are known as ebtables. They function the same way that iptables function in Linux as basic firewall rules filtering on port, ipaddress, protocol. In some cases when listening on a tap point on a network the port the interface that

is being forward to the VM is connect to a Network TAP and can only receive on side of the conversation. In this case two taps would be need one listening on each side of the conversation and the two trace would have to be reassemble much the same way Paxson did with the data collected at LBNL [Paxson 21]. Figure 4 shows an example network view of a NEMESIS running on live network data.

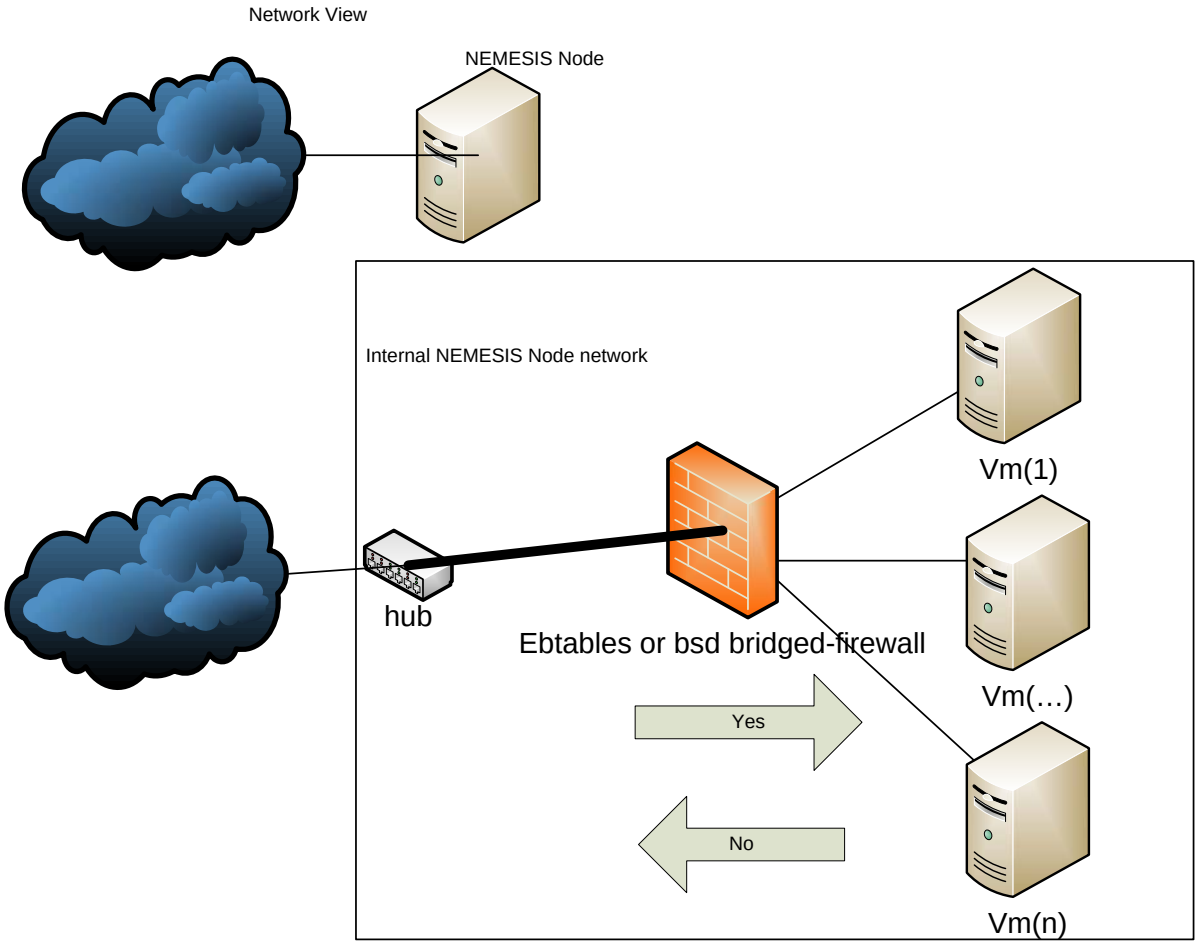

Figure 5. NEMESIS live network data view

In the instance where a network trace is present, and not live trace data. When using virtual network interface to transport recorded trace data from host to client machine one

could use TCPReplay to replay the recorded trace on the host OS directly on the virtual interface of the guest OS running the experiment. This would allow for the trace data to be recorded in advance and then feed through anonymizers, such as Crypto-PAn before being played to the VM if one is still concerned about privacy leakage. Figure 5 shows NEMESIS running on recorded sanitized network data.

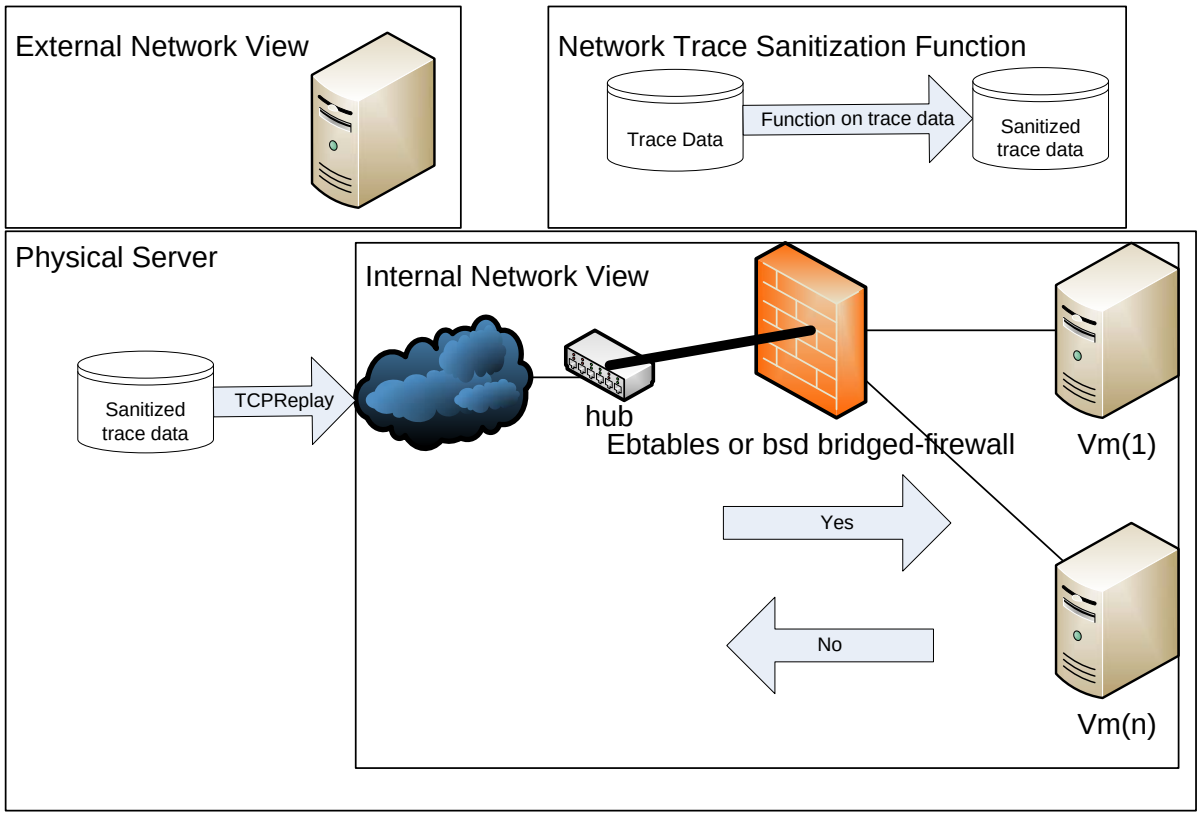

Figure 6. NEMESIS with sanitized data being replayed to the virtual machines

Problems that must be discussed with the use of virtual network interface as the primary way of moving trace data between host and guest OS for the purposes of test. When a virtual interface receives a packet a context switch is requested to handle that packet. Packets are queued however some packets are lost during context switches in observation

there was observed packet lose, however it was limited and but further testing would be need for a conclusion on using virtual interfaces in a production environment. A suggested optional replacement was a kernel module that handles the networking and queuing of packets. The second replacement option would be Ethernet-over-IP this would encapsulates the Ethernet frames in IP frames and transmit the request frame to a given guest OS. In this case a module might need to be written to filter or handle the different frames going to multiple VM on the same bridge or cloning of packets so that they could be forwarded to different experiments.

3.2.3 Policy management and enforcement

Policy management and enforcement has multiple steps. An organization that owns the trace data thus forth referred to as the owner, will have policy regarding privacy and sensitive information leaving the organization's network. An example might be that internally sending SSN is expectable but they are not to be sent out of the network. Another example might be that no information containing information about the internal network topology and sub netting scheme should be release to the general public.

Organization have developed policies on what information can leave their network and most likely will have policies on what machine can be brought on to their network, virtual or physical. The organization must also examine what information they are willing to allow a research experiment access to on their network. This might vary from a standard machine on their network, since the creator and end user of the machine is not an employee of the organization. This is where anonymization has traditionally come into play. There

would be detailed discussions on policy between the researcher and the organization on what they would be willing to expose and what they would want to have sanitized.

One of the design goals of moving the experiment to the trace owner instead of bring the trace to the experiment is that this discussion would be minimized. Instead of completely sanitizing the trace, to a point where it could have impact on the results. The policies can be enforced on the data that leaves the trace owner. The results returned from the experiment can be where the policies are enforced. It is the hope that in the future that the policy enforcement can be automated, but in this prototype it is done by human interaction by examination of the results. Trust is a common problem with all trace data collection and usage solutions. It is a problem here, if there is noise in a system than there is room for a covert channel. When the results are returned to the research sensitive information could be released in carefully crafted results. This would be an example of an active attack on the virtual infrastructure system. Much like packet injection works on anonymization solutions if one injects data into the results it is almost undetectable due to noise. However this can be minimized based on the noise in the data results that are agreed upon. If the results are a comma delimited file of alerts returned or simple numbers that represent data points, there is some noise there, however, if the results are number of events and counts, aggregate the risk could be reduced.

3.3 Process for a research experiment

This subsection outlines the process flow that a typical research would go through with an organization using this solution. The following Figure 6 is an overview that will be broken down of the process flow.

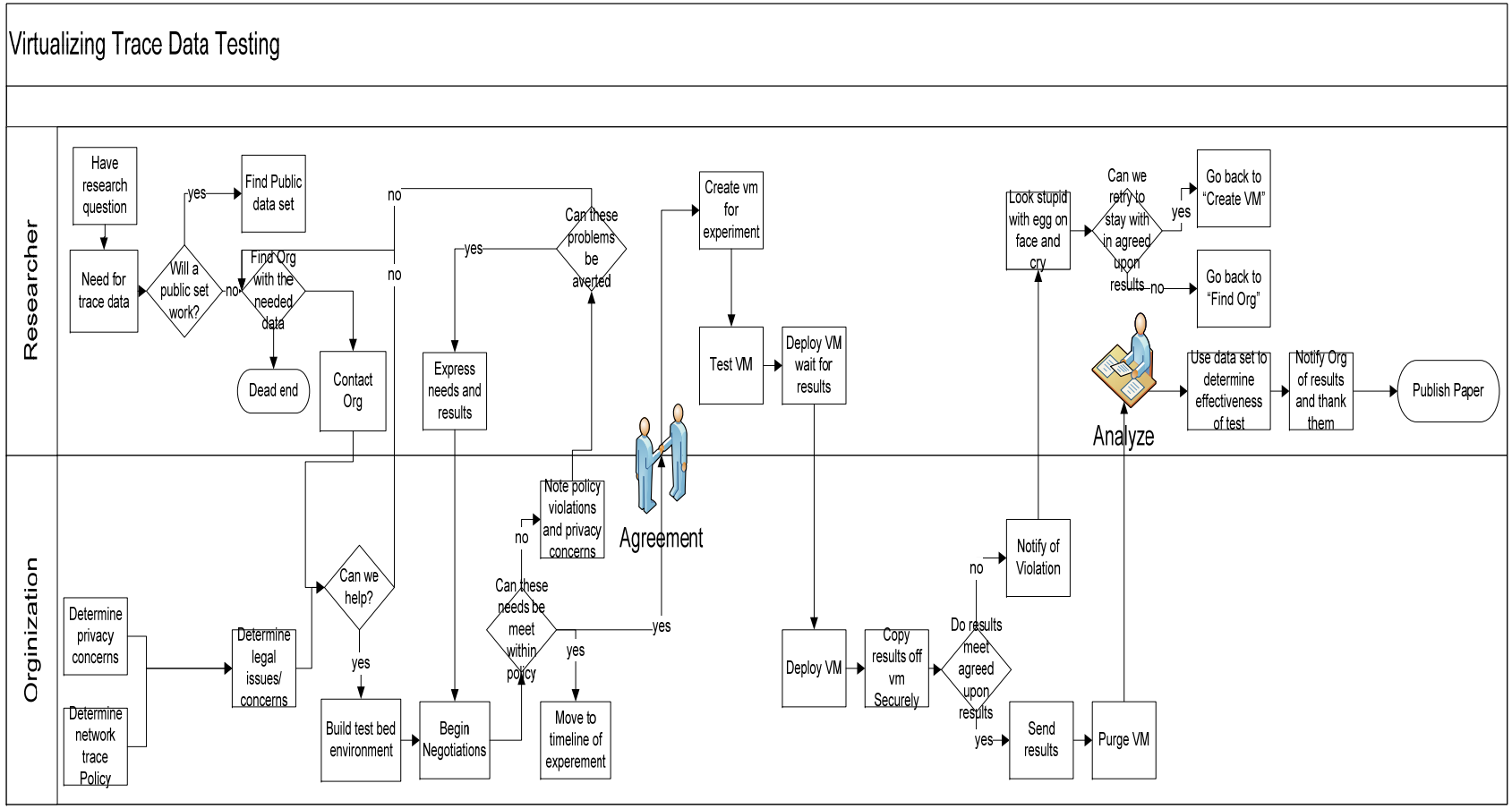

Figure 7. Process diagram

3.3.1 Initial contact and agreement

Organizations need to develop polices about network traffic. Organizations need to have security and privacy policies. Having the policy in place helps to know what do in a circumstance when an incident occurs on the network. These policies would help to govern the discussions on what data the experiments and research would be privy to for the experiments. Policies should be discussed at all levels of management and with legal taking in the concerns of the security and networking teams.

Researchers start with a problem or a question to be addressed. Once they have a question and determine that they need trace data for their solution. The data may be needed for testing or comparison. The research needs to see if a publically available data set exists that they can use, if there is a set they should do testing with that data, set while looking for an organization that has the NEMESIS node in place. Once an organization is identified the researcher would write up their needs for the data, types of data, results and research question they want to address and begin a dialogue with this information with the organization. The organization is going to need detail to determine if they can help the researcher.

Negotiations between the researchers and the organization now begin. Up until this point the organization has agreed that they could help but not the specifics of the data that they will be able to expose to the researchers. The organization knows the research problem, needs, and desires of the researchers. The organization then evaluates this against the predetermined policies, and what results they can allow out of the network. The organization must evaluate what trace and network data they can allow the researchers to see. Then they must determine what results can leave there network. What is an acceptable risk, if the

results returned can simply be aggregate data? By controlling what results leave the network the organization can make an effort to prevent future attacks on the results. When a network trace is released to the public anonymized or not there is no way to prevent against future new attacks. This is best observed by the example of Pang's attacks on the data that Paxson collect at LBNL [LBNL 21]. The discussions will go back and forth until an agreement is made or it is determined that the needs of the researcher cannot be met by the organization due to policy or data leak concerns.

3.3.2 Virtual machine creation and deployment

At this stage in the research process an agreement has been reached, and now development on the test VM can begin. While a scheduled test date has already been agreed upon the experiment VM must be built since size, traffic, result, memory, and CPU limitations will all have been outlined. These are all needed when constructing the virtual machines to understand performance needs, disk space and memory limitations so that way in the middle of the experiment the VM doesn't have to constantly be paging out memory or be pegging the CPU. For example if the VM was to use snort and it had limited memory the research may need to turn on the "lowmen" flag. The VM must also be able to complete itself write data to the results directory by the stop date, and be autonomous since it will be started and have no interaction other then booted once it leaves the researcher and is sent to the organization. Since the VM will be autonomous there will need to be extensive testing complete on the part of the research. By using all open source tools the researcher can create a test network to mimic the organization and thus minimize all integration concerns. The VM shall be sent with its start script since in some cases the start scripts can be long. Some

parts of the script will be defined by the organization, since the organization will most likely want the device to have the n graphics option enabled. This option simple removes the x display mapping of the VMs display; included in appendix B is the man pages for KVM along with all the start options.

3.3.3 Collecting results from the experiment virtual machine

Before the scheduled shutdown time for the VM it should write its results to the results segment on disk as agreed upon in the agreement stage discussed earlier. The results are then pulled off the VM and moved on to the Host OS. Once on the host OS the results can be compared against the agreed upon policies and the experiment VM and disks can be zeroed and removed. Tools can be ran on the results to detect pattern matching, whether internal network IP schemes are being leaked, raw trace data is being released through the results, unsanitized logs with raw trace data contained are being released, network vulnerabilities, basically any information other than what is agreed upon. If the agreed upon results carry a lot of noise then there would be room for a covert channel this exists in al systems, including sanitization.

If the results are found to b in violation of the agreed upon contract they are not released. At this stage it might become obvious to the organization that there was a misunderstanding or an assumption made that was to explicitly stated, and thus the results leak information that the organization does not want to leave its network. In either case the results would not be released the problems with the results sets would be brought up directly with the researcher. Some of the issues may have simply been a misunderstanding, or the results were not formatted correctly and the researcher may be given a chance to fix the VM

and the experiment rescheduled. It might be that the researcher thought that they could get a little bit extra data, in which case the organization may feel a trust violation has occurred and scrap the experiment and future work with the researcher. Another possibility is that for one reason or another the results were not written to the results section.

If the results were not written then the researcher would be notified that there was a problem with the VM and that the results were not where they were expected to be. The researcher would be given the opportunity to see if they had a similar error in there local test and given a chance to fix it and reschedule. If the VM crashed it would be rescheduled to run later when there is an opening on the server for experiments. The researcher would still be notified of the failed attempt.

If the results contain no extra data, meet policy, and are formatted as agreed upon, they are released to the researcher. The results are than analyzed and evaluated and added to the research paper which is then published.

CHAPTER 4. TEST SETUP

There has been a base system implemented with a larger test run scheduled for IT Olympics 2009 at Iowa State University. The test system includes a host server and then multiple experiment VM running at the same time. The test system host setup will be described followed by the preliminary test and then the scheduled full feature test.

4.1 Host System

The test system is a dell PowerEdge1950 with an Intel Xeone5345 quad core processors, 2 sata 7200 rpm 160 gb hard disks, 4 Gb RAM, and 3 1000/100/10 network adapters. The processor supports Intel's virtualization instructions so can be used in conjunction with KVM as a Virtual machine manager. This system will function for the tests and be able to handle all the test situations.

4.1.1 Operating system

The Operating selected was Debian based Ubuntu Server 8.10 this was choose due to implementer familiarity with the operating system. Debian is a Linux distribution that has a KVM port. The other choose would have been to use Fedora, but with the ease of use of Ubuntu and its fast growing user base, future researchers may have more familiarity with Ubuntu since it primarily used KVM and Qemu as VMM over Red Hat and Fedora using XEN, until recently. During installation the Virtualization option was not selected and KVM was not installed. If this option is not installed at installation it can be installed by "sudo apt-get install KVM." Other packages that needed to be installed are uml-tools, build-essentials, gcc, g++, python, brctl, and ebtables. Because UML-tools in Debian installs a version of

tunctl that only creates tun device and not tap devices as discussed earlier, one needs to download the source code for tunctl and build the tool. Once the tool has been built it needs mv so that it replaces the currently installed version in /sbin.

4.2 Alpha test

The Alpha Test was limited in its scope testing the core most functionality of project. The alpha test was performed during a Cyber Defense Competition at Iowa State University. The test consisted of two experiment VMs and no filtering. Due to design complications prior to the test a different test solution was used. The alternate solution involved using VM-Ware ESXi and carefully assigned virtual switches to simulate the environment of the original test system with bridged network interfaces. The challenges that were seen during this test were resolved two days after the test was ran. The challenges steamed from the default UML-tools installation of tunctl only creating tun devices and thus forcing the bridges that were in promisc mode to function as switches since the interface that were bound to the bridges were IP devices and not Ethernet devices, so only traffic bound for the virtual interface would reach the virtual interface and not all traffic.

4.2.1 Network for alpha test

The network for the alpha is similar to the network for the Full feature test only fewer systems were online for the test. Figure 7 below shows a view of the network and where the Host Os server was listening and thus where the experiment virtual machines were listening.

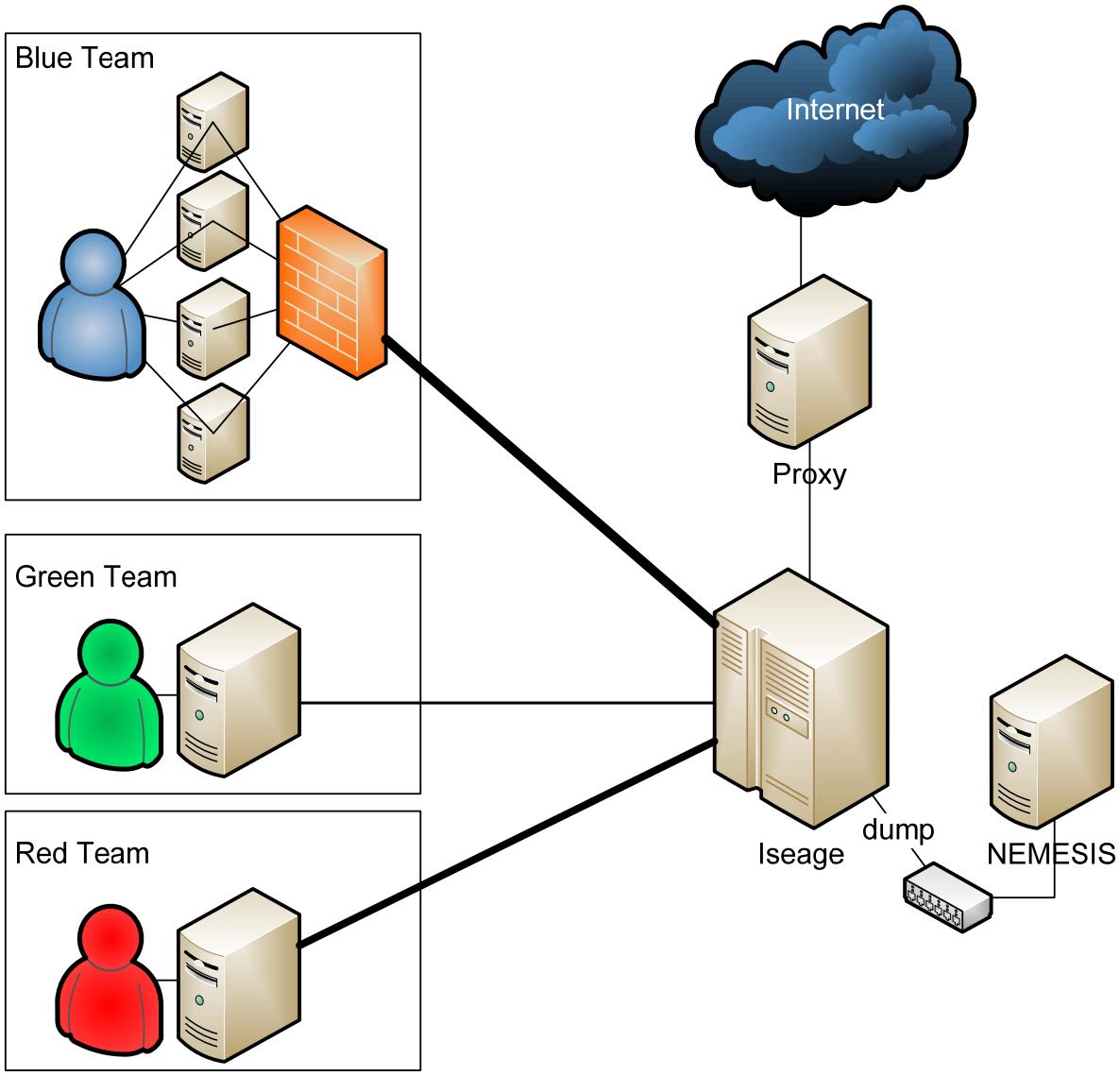

Figure 8. Alpha test network

Blue teams (defense) networks had a ftp, web, rdp, and cvs server running on the networks. There were approximately 14 blue team networks each with its own IP range and network traffic. All out bound and inbound traffic to a team's network got routed through

the ISEAGE Internet Scale Event Attack Generation Environment cluster, and then copies of those packets were dumped out to a hub in which the tap interface of the Host Server was listening on.

Red teams (attack) network consist of many different IP address ranges in which any machine connected to the network can acquire an address in any of the provided IP ranges. This is the attacker's network, where all official attacks on the Blue teams should come from. All out bound and inbound traffic to the red team network get routed through the ISEAGE cluster, to the appropriate location and then copies of those packets were dumped out to a hub in which the tap interface of the Host Server was listening on.

The Green (everyday users) network is to simulate users that access services from outside the firewall of a team. All traffic is routed through ISEAGE and a copy of the traffic is based out to the hub that the Host Server is listening on.

All Contest traffic is copied and then seen by the host server for the experiment VMs to run their tools on.

4.2.2 Virtual machines for alpha test

The Alpha test consisted of two virtual machines; each virtual machine network interface was bridged to the host machines network interface that is listening at the TAP point. There was no ebtables in place; this test was to look at the performance and packet loss and to test whether or not the traffic is seen by the VMs. So in this test it appears as if both experiment VMs are directly listening on the tap interface.

Experiment Virtual Machine 1 was designed to test the ability to run snort on a Virtual machine since during an attack multiple snort alerts should be generated. Due to the

RAM constraints of the virtual machine being set to 512 megs of RAM, the “lowmem” option of snort need to be enabled. This was a testing the ability to generate results and then copy them to a results section of the VM to be pulled off. The virtual machine operating system was Ubuntu 8.10 server with as few features installed as possible. Snort was then installed and configured start at boot and listen on the virtual machines network interface. Then the Virtual machines network device was configured statically to have no IP address and be in promisc mode. In this mode the interface listens to all inbound traffic. Then a CRON job (a command to be executed at an assigned time) was added to copy the snort log file from its default directory to /results directory on disc 30 minutes after the contest was scheduled to end. This way the researcher could follow the procedure set forth to remove the results and analyze them.

Experiment virtual machine 2 was designed to test the performance of the system and check the amount of packet lose between host network adapter and virtual machine network adapter. The contest ran for 8 hours with high and low bandwidth peaks so the packet lose percentage over that time frame would give an idea of the amount of packet lose to expect in the system. The virtual machine was a simple Ubuntu 8.10 server install. At boot the machine was scheduled to write the packet count seen by the interface to a file. Then 30 minutes after the contest was scheduled to end it was to append the packet count seen by the interface to the same file written earlier in the results section on disk.

4.2.3 Results for alpha test

Results of the experiment virtual machines in the test were varied. Virtual machine 1 results worked. The snort log detected 451 events varying from scans to web server directory

traversal. The results were written to disk and the script in appendix C was used mount the virtual disk on the host OS and then to copy the results folder from the virtual machine disk to the host OS where the results were analyzed. Virtual Machine 2 however did not work out so well. The results file showed 0 as the first entry written into the file at boot since the interface had yet to see any packets. That was the only line in the file, it never wrote the last entry into the file. It was determined the virtual machines system clock was not accurately set so that the CRON job used to write the total number of packets seen by the interface after the competition was over never ran. The lesson learned for future researcher is to check that the system clock is accurately set.

4.2.4 Test system fix for alpha test

After the contest was complete and before the network was fully taken offline the problem with the original test system setup was determined to be a custom compiled version of tunctl. After compiling from source the following WireShark screenshot was taken. Figure 8 is the WireShark image, and it shows that the problem was resolved with a source compile.

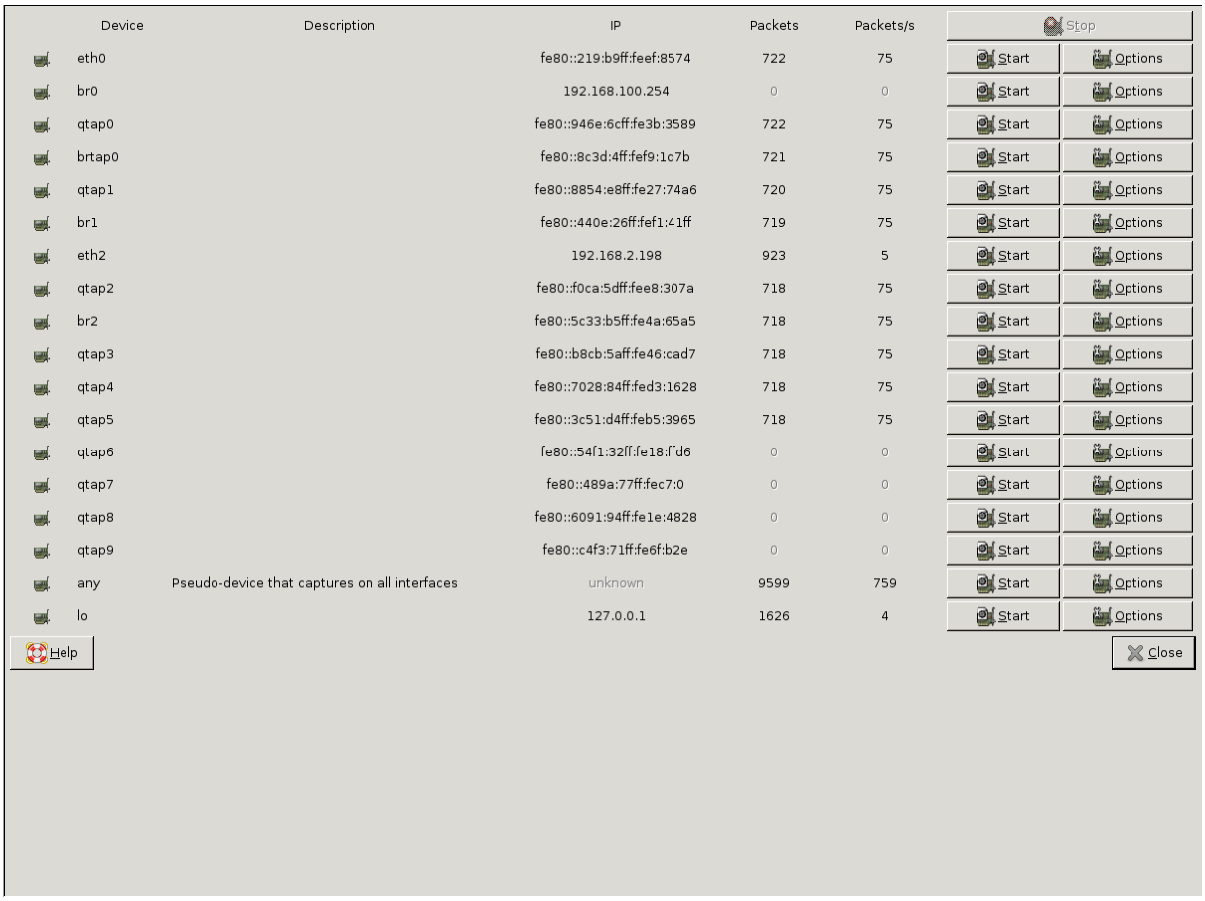

Figure 9. Alpha test WireShark fix

The wire shark image shows that traffic does travel through the virtual machines and sustains some traffic lose. The network that the WireShark image is shown in Figure 9. The network is simulated on one physical server with 4 virtual machines. The first Virtual machine is acting like a gateway to the other virtual machines. That virtual machine has bridged network connections which I represent with a hub. The VM Gateway has a bridge configured on it. Anything that is not in a VM box is configured on the physical server. Since the Wireshark image was taken while running on the host server, the interface names in Figure 7 coincide with the interface names on the physical server and not that of the interface names on the virtual machines.

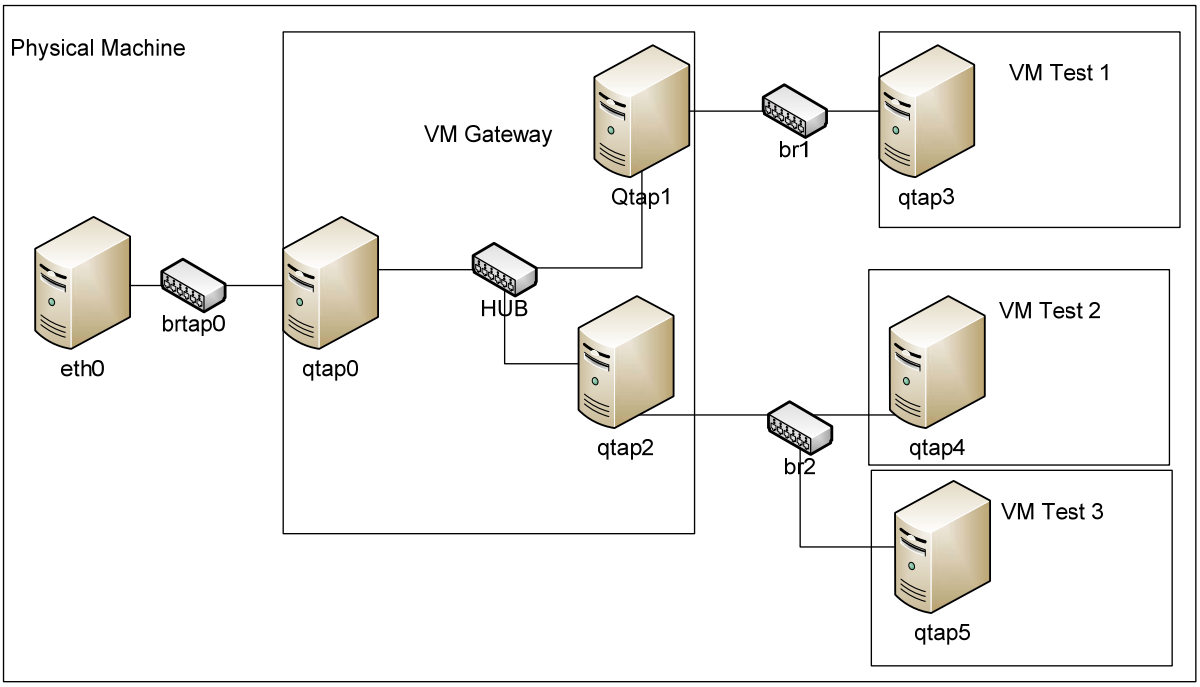

Figure 10. Virtual network configuration for Figure 6

4.3 Full feature beta test

The Beta test is to test most of the feature of the proposed system. The propose original host operating system setup will be used. The Beta test will be conducted at IT Olympics on April 20th-21st. The test shall consist of 4 virtual machines, the host box will be running a WireShark window to keep track of all packet reaching virtual interfaces. The first virtual machine shall be used a base for snort alerts. The second virtual machine shall be used with an HTTP filter and snort to test port filters in combination with a tool. The 3rd virtual machine will be counting ssh packets without a filter, the 4th virtual machine will be counting ssh packets with port 22 being filtered.

4.3.1 Network for beta test

The network for the beta is similar to the network for the alpha test only more systems will be online for the test. Figure 10 below shows a view of the network and where the Host Os server will be listening and thus where the experiment virtual machines will be listening.

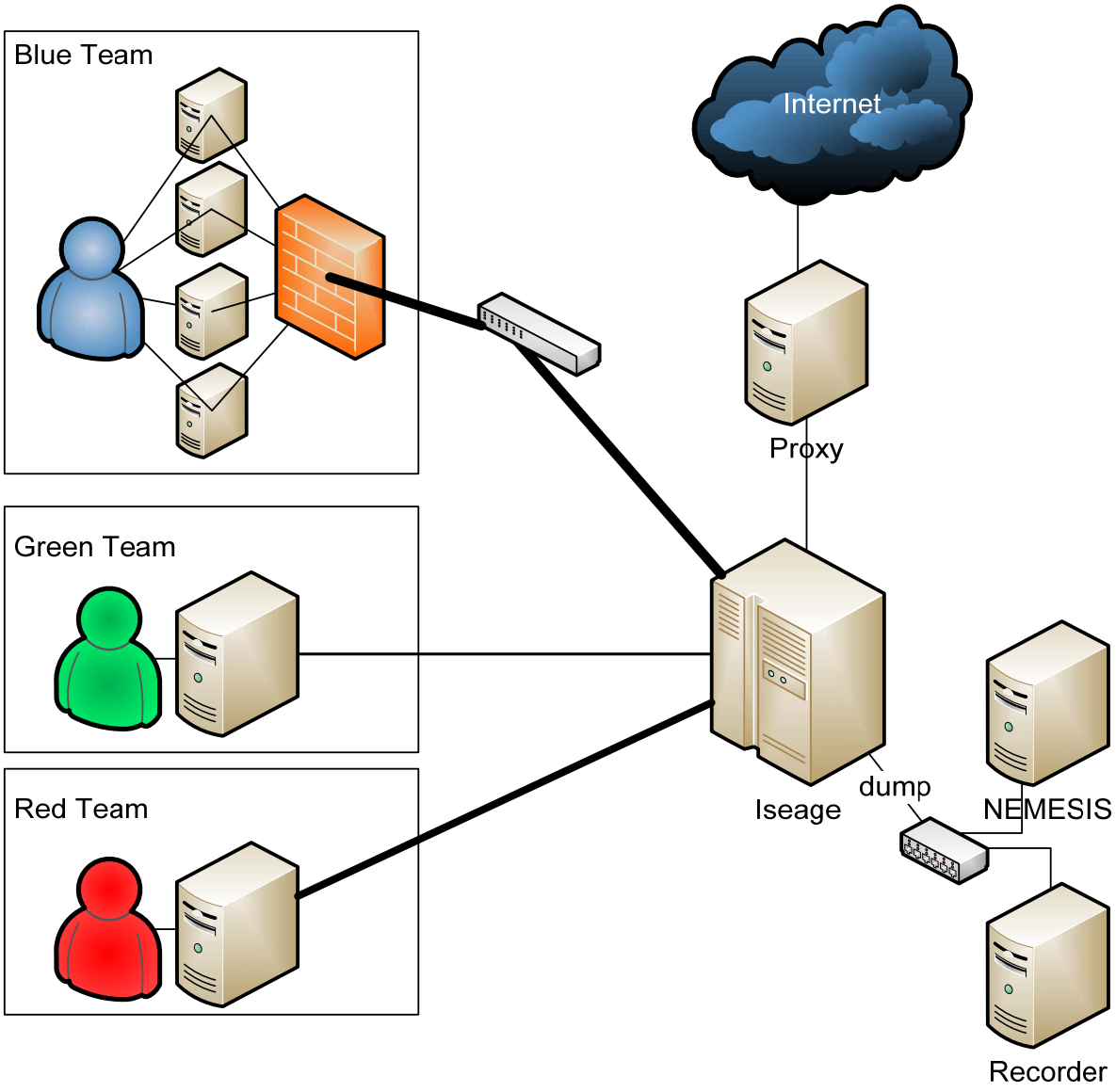

Figure 11. Beta test network

Blue teams (defense) networks will have a ftp, web, remote desktop, and cvs server running on their networks. There are approximately 14 blue team networks each with its own IP range and network traffic. All out bound and inbound traffic to a team's network get routed through the ISEAGE Internet Scale Event Attack Generation Environment cluster, and then copies of those packets were dumped out to a hub in which the tap interface of the host Server will be listening on.

Red teams (attack) network consist of many different IP address ranges in which any machine connected to the network can acquire an address in any of the provided IP ranges. This is the attacker's network, where all official attacks on the Blue teams will come from. All out bound and inbound traffic to the red team network get routed through the ISEAGE cluster, to the appropriate location and then copies of those packets were dumped out to a hub in which the tap interface of the Host Server was listening on.

The Green (everyday users) network is to simulate users that access services from outside the firewall of a team. All traffic is routed through ISEAGE and a copy of the traffic is based out to the hub that the Host Server is listening on.

All contest traffic is copied and then seen by the host server for the experiment virtual machines to run their tools on. This contest is using vlans to manage traffic which some organizations use and will test the ability to work on a network with vlans.

4.3.2Virtual machines for beta test

The Beta test consisted of four virtual machines; each virtual machine network interface was bridged to the host machines network interface that is listening at the TAP

point. There will be ebtables in place; this test is to look at the performance, packet loss, to test whether or not the traffic is seen by the virtual machines, and to see if the ebtables is an effective way to do packet filtering. So in this test it appears as if all experiment virtual machines are directly listening on the tap interface.

Experiment Virtual Machine 1 will be designed to get a baseline of results for running snort on a virtual machine during the contest. Since during the attack phase multiple snort alerts should be generated. The physical server is limited by memory constraints, so the virtual machines are limited by ram constraints. The virtual machine's ram limit will be set to 512 megabytes of RAM, the "lowmem" option of snort needs to be enabled. 20 minutes after the contest is scheduled to end the virtual machine shall copy the snort result log over to its /results directory. The virtual machine operating system will be Ubuntu 8.10 server with as few features installed as possible. Snort was then installed and configured start at boot and listen on the virtual machines network interface. Then the virtual machine's network device was configured statically to have no IP address and be in promisc mode. In this mode the interface listens to all inbound traffic.

Experiment Virtual Machine 2 was designed to compare a virtual machine with a filter in front of it to a baseline of results for running snort on a virtual machine during the contest. Since during the attack phase multiple snort alerts should be generated, this virtual machine will have a filter on the interface to prevent http traffic (port 80) from reaching the interface and thus the snort alerts generated off of port 80 will not be seen. The physical server is limited by memory constraints, so the virtual machines are limited by ram constraints. The virtual machine's ram limit will be set to 512 megabytes of RAM, the "lowmem" option of snort needs to be enabled. 20 minutes after the contest is scheduled to

end the virtual machine shall copy the snort result log over to its /results directory. The virtual machine operating system will be Ubuntu 8.10 server with as few features installed as possible. Snort was then installed and configured start at boot and listen on the virtual machines network interface. Then the virtual machine's network device was configured statically to have no IP address and be in promisc mode. In this mode the interface listens to all inbound traffic.

Experiment Virtual Machine 3 is designed to gather baseline data about the traffic that the interface sees. The virtual machine operating system will be Ubuntu 8.10 server with as few features installed as possible. The virtual machine will be running a custom libpcap application designed to keep track of the total number of Ethernet, ARP, ICMP, http, and SSH, packets seen. SSH and HTTP will be monitored by the looking for source and destination ports of 22 or 80 respectively. The application will be configured to start at boot and listen on the virtual machines network interface. Then the Virtual machine's network device will be configured statically to have no IP address and be in promisc mode. In this mode the interface listens to all inbound traffic.

Experiment Virtual Machine 4 is designed to gather data about the traffic that the interface sees, with a filter in place. The virtual machine operating system will be Ubuntu 8.10 server with as few features installed as possible. The virtual machine will be running a custom libpcap application designed to keep track of the total number of Ethernet, ARP, ICMP, http, and SSH, packets seen. SSH and HTTP will be monitored by the looking for source and destination ports of 22 or 80 respectively. The application will be configured to start at boot and listen on the virtual machines network interface. There will be a filter in place to block SSH (port 22) traffic from reaching the virtual machine. Then the virtual

machine's network device will be configured statically to have no IP address and be in promisc mode. In this mode the interface listens to all inbound traffic.

4.3.3 Traffic recorder

In addition to the NEMESIS Node, another machine will be listening at the same point as the NEMESIS node to record all network traffic seen. This is so that a test can be conducted on the use of TCPReplay to replay the traffic back to the virtual interfaces, and see if the same results are achieved.

CHAPTER 5. CONCLUSION

NEMESIS is a system level approach to collecting and using private trace data that has been historically hard to gain access to. Nemesis is a framework that allows researchers to build their experiment with the tools they need, and with the only restrictions being, disk, processor and ram usage. Nemesis allows researchers to build their experiments it also allows for policy to be enforced by the trace data owner. The policy can be enforced on the type and quality of data that the researcher's experiment virtual machine sees. The policy also comes in to play on what data and results the researcher is allowed to take out of the network. The system level approach allows for a security n depth model.

5.1 Successes

Using a virtual machine as the experiment benefits the researcher and the organization. The researcher benefits because they can use multiple tools running on their experiment virtual machine and correlate the results as opposed to previous solutions that require the researchers to quarry a database, and or write their applications in a custom programming language. The organization benefits from a security in depth model instead of a basic on layer of anonymization. Trace data anonymization can be seen as security through obscurity which has long been seen as a bad approach, in anonymization of trace data, Pang proved that this was the fact when he reversed the network topology [Pang 4]. Anonymization of trace data can be seen as one layer to help protect the organization private information. The NEMESIS solution allows for firewall rules to prevent some traffic from reaching the virtual machines, it also allows for some control over what results leave the

organization, to help prevent against future attacks and network topology leaks. The use of virtual machines also means that the experiments can be restricted from the network as a whole and be isolated, so that they can not affect the network. The system allows for most of Saltzer's and Schroder's design principles.

Principle of least privilege is the principle that says that objects and subjects should have sufficient privileges to do what they need to do and nothing more. When using virtual machines the experiment runs as a virtual machine, but the user that launches the virtual machines determines the access rights of the virtual machine on the host machine. This allows for the ability to restrict access and privileges on the host OS. Principle of fail-safe defaults means deny all by default and allow rights and privileges as needed. In the case of the virtual machines they need privileges to /dev/tun, nut do not need access to a lot of other host OS devices. Principle of economy of mechanism is reached by having a simple virtual machine framework that experiments are just deployed to and then the solution uses the existing network devices to handle network traffic to prevent over complicating things. Principle of open design is achieved by publishing the frame work and using open source software in the implementation. Principle of separation of privilege is achieved by the policy management system to restrict the results that leave along with the traffic that it sees. Principle of least common mechanism depending on virtual machine implementation this can be achieved. If the virtual machines' disks are held on a cloud style NAS then the disk arrays and read covert channel is removed, if there is enough RAM in the machine then that will be removed, but there is always some common mechanisms like hardware buffers that are always shared. Principle of psychological acceptability is achieved due to fact that researchers have the ability to build a VM and have control over the disk image of the

experiment and the organization has control over the hardware and results that leave there network.

5.2 Limitations

This Solution is not without its limitations. Currently the filtering is not capable to do deep packet inspection on packets and filter based on that, however with more work this may be possible.

Currently the policy management of results is checked by hand and not by an automated system. While using one way network taps, the system would need to join the network traces before replaying it on to the interfaces. Virtual machines are limited by size, RAM, and CPU, so the total number and what is doable with the virtual machines are determined by how much of the physical assets are allocated to the virtual machines.

Another limitation for the system is that the trace data is not published, so every time the researcher wants to run another experiment has to reach an agreement with the organization in order to gain access to the data again. If the researcher is listening to a live network feed then he cannot reproduce the test since the trace is not recorded.

CHAPTER 6. FUTURE WORK

There are areas that could benefit from future work. The system needs to be tested with recorded trace data replayed over the virtual interface. The system needs to be tested with recorded anonymized data played over the virtual interface.

The Policy management piece of the system could be expanded and fully implemented so that it can be automated, including anonymizing data sets and replaying them to a given virtual interface, and filtering those data sets based on the policies set forth on what the experiment virtual machine can see.

Some of the networking utilities might benefit from a custom kernel module that is different than a bridge; this would work with the policy manager so that the data sets might be pushed to two virtual interfaces at the same time.

Since in the case of the test the researcher and trace owner was the same the policy negotiations were not test and need to be tested.

APPENDIX A. BRCTL HELP

brctl –addbr <bridgename> -- creates a bridge interface

brctl –delbr <bridgename> -- deletes a bridge interface

brctl –addif <bridgename> <interface name>-- adds an interface to a bridge interface

brctl –delif <bridgename> <interface name>-- deletes an interface to a bridge interface

revised

APPENDIX B. KVM HELP

The following is an excerpt from the man page of QEMU and are used in the test system for more information please view the full man page written by Fabrice Bellard.

You can connect a CDROM to the slave of ide0:

qemu -drive file=file,if=ide,index=1,media=cdrom

If you don't specify the "file=" argument, you define an empty drive:

qemu -drive if=ide,index=1,media=cdrom

By default, interface is "ide" and index is automatically incremented:

qemu -drive file=a -drive file=b"

is interpreted like:

qemu -hda a -hdb b

-boot [a|c|d|n]
Boot on floppy (a), hard disk (c), CD-ROM (d), or Etherboot (n). Hard disk boot is the default.

-localtime
Set the real time clock to local time (the default is to UTC time). This option is needed to have correct date in MS-DOS or Windows.

Display options:

-nographic
Normally, QEMU uses SDL to display the VGA output. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console. Therefore, you can still use QEMU to debug a Linux kernel with a serial console.

-vnc display[,option[,option[,...]]]
Normally, QEMU uses SDL to display the VGA output. With this option, you can have QEMU listen on VNC display display and redirect the VGA display over the VNC session. It is very useful

to enable the usb tablet device when using this option (option -usbdevice tablet). When using the VNC display, you must use the -k parameter to set the keyboard layout if you are not using en-us. Valid syntax for the display is

Network options:

-net nic[,vlan=n][,macaddr=addr][,model=type]
Create a new Network Interface Card and connect it to VLAN n (n = 0 is the default). The NIC is an rtl8139 by default on the PC target. Optionally, the MAC address can be changed. If no -net option is specified, a single NIC is created. Qemu can emulate several different models of network card. Valid values for type are "i82551", "i82557b", "i82559er", "ne2k_pci", "ne2k_isa", "pcnet", "rtl8139", "e1000", "smc91c111", "lance", "mcf_fec" and "usb". Not all devices are supported on all targets. Use -net nic,model=? for a list of available devices for your target.

-net tap[,vlan=n][,fd=h][,ifname=name][,script=file]
Connect the host TAP network interface name to VLAN n and use the network script file to configure it. The default network script is /etc/qemu-ifup. Use script=no to disable script execution. If name is not provided, the OS automatically provides one. fd=h can be used to specify the handle of an already opened host TAP interface. Example:

qemu linux.img -net nic -net tap

APPENDIX C. MOUNT SCRIPT

mount –t ext3 –o loop, offset=32256 /path/to/image /mnt/point

cp /mnt/point/results results/vm#

Very basic setup guide

1) Install OS (in guide ubuntu server 8.10)
2) In installation select only to have ssh server option not virtualization.
3) When done installing verify you can ssh into the server, go to a location and ssh.
4) Run commands
    a. sudo apt-get update
    b. sudo apt-get upgrade
    c. sudo apt-get install kvm xserver-xorg-core uml-utilities ebtables gcc g++ python build essential vncviewer gcj ethtool wireshark kpartx
5) verify that you can use x-forwarding over ssh if you are ssh in.
6) kvm-img create –f raw diskname.img size{numberM, number G}
7) recompile tunctl from source and copy into /usr/sbin/

    http://sourceforge.net/project/showfiles.php?group_id=233549
8) sample network creation deletion script customize for your bridge needs
    a. Script

    #!/bin/bash

```
# id of the user running qemu (kvm)
USERID=0000

# number of TUN/TAP devices to setup
NUM_OF_DEVICES=5

case $1 in
      start)
            modprobe tun
              ifconfig eth1 0.0.0.0
            echo -n "Setting up bridge device br0"
            brctl addbr br0
            ifconfig br0 192.168.100.254 netmask 255.255.255.0 up
            for ((i=0; i < NUM_OF_DEVICES ; i++)); do
                  echo -n "Setting up "
                  tunctl -p -b -u $USERID -t qtap$i
                  ifconfig qtap$i 0.0.0.0 promisc up
            done
              brctl addif br0 qtap0
##################
#  tap
##################
              brctl addbr brtap0
            ifconfig brtap0 up 0.0.0.0 promisc
              brctl addif brtap0 eth1
              brctl addif brtap0 qtap1
              brctl addif brtap0 qtap2
              brctl addif brtap0 qtap3
              brctl addif brtap0 qtap4
      ;;
      stop)
            for ((i=0; i < NUM_OF_DEVICES ; i++)); do
                  ifconfig qtap$i down
#                  brctl delif br0 qtap$i
                  tunctl -d qtap$i
            done
            ifconfig br0 down
              brctl delif br0 qtap0
            brctl delbr br0
##################
#  tap
##################
              ifconfig brtap0 down
              brctl delif brtap0 eth1
              brctl delif brtap0 qtap1
```

```
                brctl delif brtap0 qtap2
                brctl delif brtap0 qtap3
                brctl delif brtap0 qtap4
                brctl delbr brtap0

        ;;
        *)
                echo "Usage: $(basename $0) (start|stop)"
        ;;
esac
```

9) Download the iso of the os you want to install

10) Build image kvm start command

    a. kvm –hda /path/to/disk –cdrom /path/to/iso/or/cdrom/device –m amount \

       –boot d –net nic, macaddr=SomeValidMac. Model=e1000 \

       –net tap, ifname=NetworkInterfaceName, script=no –localtime

11) Start image for setup /develop image

    a. kvm –hda /path/to/disk –m amount – boot c \

       –net nic, macaddr=SameValidMac. Model=e1000 \

       –net tap, ifname=NetworkInterfaceName, script=no –localtime

12) Start for deploy

    a. kvm -vnc none –hda /path/to/disk –m amount – boot c \

       –net nic, macaddr=SameValidMac. Model=e1000 \

       –net tap, ifname=NetworkInterfaceName, script=no –localtime

13) mount –t ext3 –o loop, offset=32256 /path/to/image /mnt/point

    cp /mnt/point/results results/vm#

BIBLIOGRAPHY

[1] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces. In Proceedings of the Network and Distributed System Security Symposium, February 2007.

[2] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In Proceedings of the IEEE Symposium on Security and Privacy, 2002.

[3] S. Coull, M.P. Collins, C.V. Wright, F. Monrose, and M. Reiter. On Web Browsing Privacy in Anonymized NetFlows. In Proceedings of the USENIX Security Symposium, August 2007.

[4] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The devil and packet trace anonymization. ACM SIGCOMM Computer Communications Review, 36(1):29—38, 2006.

[5] J C Mogul and M Arlitt. Sc2d: An alternative to trace anonymization. In Proceedings of the SIGCOMM 2006 Workshop on Mining Network Data, 2006.

[6] J. Xu, J. Fan, M. H. Ammar, , and S. B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In Proceedings of the IEEE International Conference on Network Protocols, 2002.

[7] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In Proceedings of the 22nd IEEE International Conference on Data Engineering, 2006.

[8] Greg Minshall. tcpdpriv tool. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[9] Jinliang Fan, Jun Xu, Mostafa H. Ammar Cryptography-based Prefix-preserving Anonymization http://www.cc.gatech.edu/computing/Telecomm/projects/cryptopan/.

[10] Ruoming Pang and Vern Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In Proceedings of ACM SIGCOMM, 2003.

[11] D. Koukis, S. Antonatos, and K. Anagnostakis. On the privacy risks of publishing anonymized ip network traces. In Proceedings of Communications and Multimedia Security, pages 22–32, October 2006.

[12] D. Koukis, S. Antonatos, D. Antoniades, P. Trimintzios, and E.P. Markatos. A generic anonymization framework for network traffic. In Proceedings of the IEEE International Conference on Communications (ICC 2006), June 2006.

[13] Jelena Mirkovic. Privacy-safe network trace sharing via secure queries. In NDA‘ 08: Proceedings of the 1st ACM workshop on Network data anonymization, pages 3–10, New York, NY, USA, 2008. ACM.

[14] T. Brekne, A. Arnes, and A. Sleb. Anonymization of ip traffic monitoring data attacks on two prefix-preserving anonymization schemes and some proposed remedies. In Proceedings of the Workshop on Privacy Enhancing Technologies, page 179196, May 2005.

[15] Lobster web page. http://www.ist-lobster.org/publications/deliverables/D1.1a.pdf.

[16] TCPDump web page. http://www.tcpdump.org/.

[17] Tunctl web page. http://tunctl.sourceforge.net/.

[18] Lennert Buytenhek Brctl man page http://www.linuxcommand.org/man_pages/brctl8.html

[19] KVM web page http://www.linux-kvm.org/page/Main_Page

[20] Fabrice Bellard KVM man page http://linux.die.net/man/1/qemu-kvm

[21] LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing/.

[22] Lennert Buytenhek brctl man page

ACKNOWLEDGEMENTS

I would like to thank my wife first and foremost for everything, from late nights to offloading of stress that I have put her through while writing this. I would like to thank the friends that I used as sounding boards. I would like to thank Doug Jacobson and Steve Russell for the support they should me throughout my college career. Finally Thomas Daniels my major professor deserves a lot of credit for all of the help and guidance he has shown me. He has help with the idea, was a sounding board for me throughout college, and listened when I was stressed.