Cooperative ReVirt: Adapting Message Logging for Intrusion Analysis

Murtaza Basrai and Peter M. Chen

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
University of Michigan
http://www.eecs.umich.edu/CoVirt

Abstract: Virtual-machine logging and replay enables system administrators to analyze intrusions more completely and with greater integrity than traditional system loggers. One challenge in these types of systems is the need to log a potentially large volume of network traffic. Cooperative ReVirt adds message-logging techniques to ReVirt to reduce the amount of network traffic that needs to be logged. Cooperative ReVirt adapts message-logging techniques to address the challenges of intrusion analysis, such as the need to work in the presence of network attacks and unreliable networks, the need to support asymmetric trust relationships among computers, and the need to support dynamic trust and traffic patterns. Cooperative ReVirt is able to reduce the log volume needed to replay a computer by an average of 70% for a variety of distributed computing benchmarks, while adding less than 7% overhead. Measurements of a live network indicate that Cooperative ReVirt would be able to avoid logging 85% of the received network data.

1. Introduction

Making computers perfectly secure appears to be unachievable, at least in the short term. The steady stream of security alerts, patches, and incidents over the past few years indicates that computer break-ins will be with us for the foreseeable future. Given this, an important component of defensive strategy is to analyze attacks after they occur. Post-attack analysis is used to understand an attack, fix the vulnerability that allowed the compromise, and repair any damage caused by the intruder.

Most computer systems enable some analysis of intrusions by logging various events [Anderson80], such as login attempts, TCP connection requests, and web server logs. Unfortunately, these types of audit logs are inadequate in terms of integrity and completeness.

Current system loggers lack integrity because they assume the operating system kernel is trustworthy—a bad assumption given the size, complexity, and track record of modern operating systems. Attackers who compromise the operating system can forge misleading log records or prevent useful log records from being saved after they compromise the operating system; they may even be able to delete log records that were written before the point of compromise. The absence of useful log records after the point of compromise makes it very difficult to assess and fix the damage incurred in the attack.

Current system loggers lack completeness because they do not log sufficient information to recreate or understand all attacks. Typical loggers save only a few types of system events, and these events are often insufficient to determine with certainty how the break-in occurred or what damage was inflicted after the break-in. Instead, the administrator is left to guess what might have happened, and this is a painful and uncertain task.

A recently developed system called ReVirt addresses the problems in current system loggers [Dunlap02]. To improve the integrity of the logger, ReVirt encapsulates the target system (both the operating system and the applications) inside a virtual machine, then places the logging software beneath this virtual machine. Running the logger in a different domain than the target system protects the logger from a compromised application or operating system. ReVirt continues to log the actions of intruders even if they replace the target boot block or the target kernel.

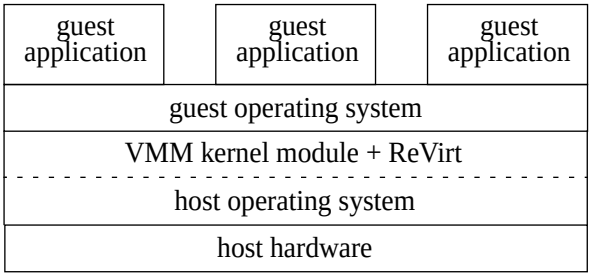


Figure 1: FAUmachine virtual machine and ReVirt replay service. ReVirt is a virtual-machine replay service. Our version of the FAUmachine virtual machine is implemented as a loadable kernel module in the host Linux operating system. The device and interrupt drivers in the guest operating system use host services such as system calls and signals.

To improve the completeness of the logger, ReVirt uses checkpointing and logging techniques to enable replay-driven intrusion analysis. Rather than provide the system analyst with ad hoc, partial information, ReVirt is able to replay the complete, instruction-by-instruction execution of the virtual machine, even if that execution depends on non-deterministic events such as interrupts and user input. An administrator can use this type of replay to answer arbitrarily detailed questions about what transpired before, during, and after an attack. For example, because ReVirt can replay instruction-by-instruction sequences, the administrator can see the complete state of registers, memories, and disk drives on arbitrary instruction boundaries.

To enable replay-driven intrusion analysis, ReVirt logs all non-deterministic events (viz. external input and interrupt timing), the largest component of which is incoming network data. This paper seeks to apply message-logging techniques to reduce the amount of data that ReVirt needs to log. We call the resulting system Cooperative ReVirt, because it leverages cooperation between communicating computers to reduce the size of ReVirt's log. Cooperative ReVirt reduces the size of ReVirt's log by 70% for a variety of distributed system benchmarks while adding less than 7% overhead. Measurements of a live network indicate that Cooperative ReVirt would be able to avoid logging 85% of the received network data.

The rest of the paper is organized as follows. Section 2 presents an overview of the virtual machine and logging functionality used by ReVirt. Sections 3 and 4 describe how Cooperative ReVirt leverages message-logging techniques to reduce the amount of data that ReVirt must log. Section 5 evaluates Cooperative ReVirt in terms of the log space it saves and the time overhead it adds. Section 6 discusses related work, and Section 7 concludes.

2. ReVirt

2.1. Virtual machine

ReVirt is a replay service that works in the context of a virtual machine monitor (Figure 1). A virtual-machine monitor (VMM) is a layer of software that emulates faithfully the hardware of a complete computer system [Goldberg74]. The abstraction created by the VMM is called a virtual machine. The main benefit of implementing the replay service in the VMM is the small size of the VMM. A VMM is several orders of magnitude smaller than a normal operating system and thus makes a better trusted computing base.

The current ReVirt prototype uses a virtual machine called FAUmachine (formerly called UMLinux) [Buchacker01]. In FAUmachine, the host platform on which the VMM runs is another operating system, which we refer to as the *host* operating system to distinguish it from the *guest* operating system that runs inside the virtual machine.

FAUmachine provides a software analog to each peripheral device in a normal computer system. Table 1 shows the mapping from each host component or event to its software analog in the virtual

Host component or event	Emulation mechanism in FAUmachine
hard disk	host raw partition
CD-ROM	host /dev/cdrom
floppy disk	host /dev/floppy
network card	TUN/TAP virtual Ethernet device
console	host stdout
video card	none (display to remote X server)
current privilege level	VMM variable
system calls	SIGUSR1 signal
timer interrupts	timer + SIGALRM signal
I/O device interrupts	SIGIO signal
memory exception	SEGV signal
enable/disable interrupts	mask signals

Table 1: Mapping between host components and FAUmachine equivalents.

machine. The most relevant peripheral for this paper is the network card. FAUmachine emulates the network card with a TUN/TAP virtual Ethernet card in Linux. The guest operating system sends Ethernet frames to the TUN/TAP driver in the host operating system, which then routes it to the intended receiver like any other packet. Similarly, if another computer sends a packet to the guest operating system's IP address, the host operating system receives this and forwards it to the TUN/TAP driver, which can then be received by the guest operating system.

2.2. Logging and replay

To enable replay of the original execution, ReVirt must checkpoint the state of the virtual machine, then log all non-deterministic events that affect the virtual machine's execution. The checkpointed state includes registers, memory, and disk; it does not include the processor's microarchitectural state (e.g., pipeline state), as this state does not affect the software running on the virtual machine. The non-deterministic events that must be logged are keyboard input, network input, and interrupt timing. Note that disk input need not be logged. Because the disk is checkpointed and disk writes are replayed, disk reads will return the same data as during the original run. Interrupt timing is the most difficult event to replay. Interrupts are emulated by software signals, and these must be replayed at the exact same instruction as during the original run [Bressoud96, Slye98]. ReVirt uses the x86 performance counters to count the number of branches. The combination of branch count and the address of the interrupted instruction uniquely identifies the point in the instruction stream where the signal was delivered.

ReVirt has been shown to add reasonable time and space overhead. The main time overhead is due to virtualization, which adds 14-35% in run time for a variety of benchmarks such as SPECweb99 and a local kernel build [King03]. Virtualization overhead will vary with the specific virtual machine platform; for example, the Xen virtual machine [Barham03] slows the machine by only a few percent. The time overhead for logging is nominal for these benchmarks, ranging from 4-8% [Dunlap02]. Space overhead of log-

ging is minimal for local applications. For example a local kernel build added 80 MB/day of log space. However, log volume can be much greater for network-intensive workloads. Building the kernel over a network file system in the system measured by [Dunlap02] generated 1.2 GB/day of log volume on the client, and SPECweb99 generated 1.4 GB/day of log volume on the web server. At current prices and sizes, modern disks can store this volume of log data at reasonable cost; however worst-case scenarios are more troubling. If a computer received an intensive message stream from a computer at LAN speeds (say, 100 Mb/second), the log could grow at a rate of 1 TB/day! This rate of log growth would strain both the storage capacity and bandwidth of modern disks.

The purpose of this paper is to reduce the volume of data that must be logged by ReVirt. Ideally, Cooperative ReVirt would reduce the log volume from LAN speeds to WAN speeds.

3. Expanding the unit of replay

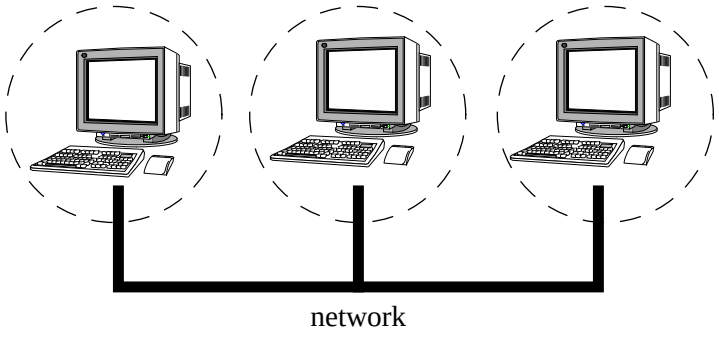
Replay systems such as ReVirt can be viewed as a perimeter around a unit of replay. This unit of replay can be chosen arbitrarily, as long as several conditions are met. First, the state of the unit of replay must be checkpointed. Second, the input coming into the unit of replay from outside the perimeter must be logged. Third, the unit of replay must execute deterministically with respect to any state or actions visible to an external observer.

In ReVirt, the perimeter is drawn around a single virtual machine (Figure 2a), so ReVirt must log all incoming messages. The volume of logged data can be reduced by expanding the unit of replay to include several computers (Figure 2b). We use the term “replay set” to describe the set of computers that are enclosed as a single replay unit. Logging a cluster of computers as a single unit of replay always generates less log data than logging each computer individually (input from outside the cluster must be logged in either case, whereas messages between machines in the cluster need only be logged if the unit of replay is individual computers). In either case, a checkpoint must include the state of all computers in the cluster. Even if one is only interested in replaying a single computer, it may still save log volume to expand the unit of replay to include several computers, depending on whether the input into the extra computers (which now must be logged) outweighs the messages from the extra computers to the original computer (which now need not be logged).

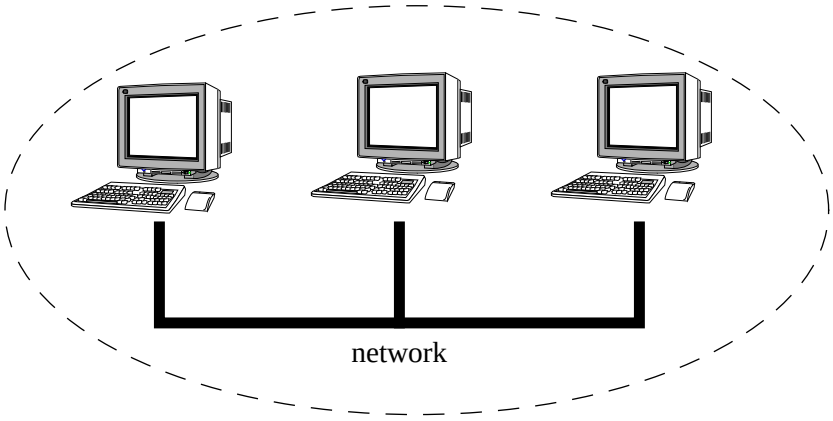
As a specific example, imagine that an administrator seeks to provide replay capability to all computers on a LAN. Logging and replaying each computer requires each computer to log messages from other computers on the LAN (as well as messages from computers outside the LAN). The maximum rate of the network message portion of the log on such a configuration is equal to the aggregate bandwidth between all machines on the LAN. If she expands the unit of replay to include all computers on the LAN, the logging system no longer needs to log messages between computers on the LAN, and the maximum rate of the network message portion of the log drops to the bandwidth coming from outside the LAN, which is likely orders of magnitude less than the aggregate LAN bandwidth. Of course, there are limits to how large the unit of replay can be. For instance, computers from different administrative domains (such as a home machine and an e-commerce web server) cannot form a single unit of replay.

There are many considerations when deciding which computers to include in the unit of replay. As discussed above, the main advantage of adding a computer to the unit of replay (rather than replaying the computer separately) is the reduction in log volume. The main disadvantage is the need for both computers to cooperate in order to replay an execution for intrusion analysis. This disadvantage can be ameliorated by conducting the replay on non-production computers or cloned virtual machines, but replaying a cooperating set of computers is still more complex than replaying a single computer. A side effect of requiring cooperation in replay is an increased level of vulnerability. If the logging system on any computer within the unit of replay is compromised, then it will not be possible to replay any computer in the replay set that received a message from the one with the compromised logging system.

These disadvantages imply that multiple computers should be included in a single unit of replay only if they communicate frequently and are part of the same trust domain. Good computers to include in a cli-



(a) Three separate replay units



(b) One combined replay unit

Figure 2: Units of replay. The original ReVirt system assumed each computer would replay as an individual unit, which forced each computer to log all incoming messages. Expanding the unit of replay to encompass multiple computers reduces the volume of data that must be logged to support replay. The disadvantage to expanding the unit of replay in this manner is an increased trust of (and therefore an increased vulnerability to errors in) other computers in order to analyze intrusions.

ent's replay set are the servers that the client communicates with frequently, such as the client's mail server, file server, and web server.

4. Design and Implementation of Cooperative ReVirt

Cooperative ReVirt leverages the above reasoning to reduce the total volume of logging needed to replay a set of computers. This section describes various aspects of the design of Cooperative ReVirt. The first two aspects of the design (coordinated checkpointing and message ordering) have been explored in traditional applications of message-logging techniques. The last three aspects (tolerating network attacks, asymmetric trust, and dynamic replay sets) are new issues that arise because we are applying message-logging techniques for the new purpose of intrusion analysis.

4.1. Checkpointing a multi-computer replay unit

Because Cooperative ReVirt replays a set of computers, it must start from a checkpoint of that set of computers. There are numerous ways to handle this issue, including coordinated checkpointing, uncoordinated checkpointing, and communication-induced checkpointing [Elnozahy02]. Our goal was to focus on the new issues that arise for intrusion analysis, so we chose to use the simplest strategy, which was to combine sender-based logging with a two-phase coordinated checkpoint [Elnozahy94]. We envisage a usage model in which periodic coordinated checkpoints are taken to bound the time period over which replay for intrusion analysis takes place.

4.2. Tolerating unreliable networks

A common issue that arises for message-logging systems is how to deal with unreliable networks. Networks can change a sequence of messages by re-ordering them, duplicating them, dropping them, or corrupting them. These changes lead to differences between the message sequence received during logging and the message sequence received during replay. We address this issue by adding a reliable communication protocol under Cooperative ReVirt [Johnson89]. Rather than building a custom reliable communication protocol, we take the Ethernet frames that are sent by FAUmachine (over the virtual Ethernet device) and send them over a TCP stream to the receiving computer (Figure 3). A proxy process on the receiving computer then unpacks the Ethernet frames and sends them to the receiving virtual machine's TUN/TAP device. During replay, the receiving computer's proxy provides the data to the replay manager via a pipe.

4.3. Tolerating network attacks

Applying message-logging techniques to intrusion analysis raises several issues that do not exist for the traditional fault-tolerance uses of message logging. One of those issues is the need to defend against attackers trying to mislead the replay system into using a different message stream during replay than it used during the logging run. While sending the Ethernet frames over TCP protects the system against accidental changes to the message stream, it does not defend against a malicious modification of the message stream.

Cooperative ReVirt uses two techniques to defend against network attacks. First, we add a cryptographic hash to each Ethernet frame to defend against modification of the source address and payload inside the Ethernet frame. To compute this cryptographic hash, we compute the SHA-1 hash of the Ethernet header and payload, then encrypt the resulting hash value using a symmetric key encryption algorithm (AES). The symmetric key is unique to each pair of computers and is specified in our system in a configuration file. An alternative solution would have been to use the IPsec-AH patch to the Linux operating system on the host (adding our own hash turned out to be easier). This cryptographic hash protects both the data being sent and the source address. Protecting the source address is important because the receiver must be able to reliably identify if the sender is in its replay set and, if so, which computer sent the message. If an attacker were able to fool the receiver into thinking the message was sent from a cooperating computer, the receiver would neglect to log a message it needed for replay.

Second, we add a sequence number to the Ethernet frame to defend against replay attacks (note that a replay attack is different from the replay service that Cooperative ReVirt is trying to provide). The sequence number increases monotonically for messages for each (sender, receiver) pair. The sequence number that we add must also be included in the data protected by the cryptographic hash, otherwise the attacker could simply manufacture a new sequence number for an old message.

4.4. Asymmetric trust

A second consideration that arises in the context of intrusion analysis is the issue of trust between replaying computers. With traditional message logging, all computers in the system trust each other as they jointly perform rollback-recovery. In contrast, with intrusion analysis, not all computers in the system trust each other. Further, it is likely that even if computer A trusts computer B, computer B may not trust computer A. For example, a client may be willing to depend on its file server to enable replay, but the file server

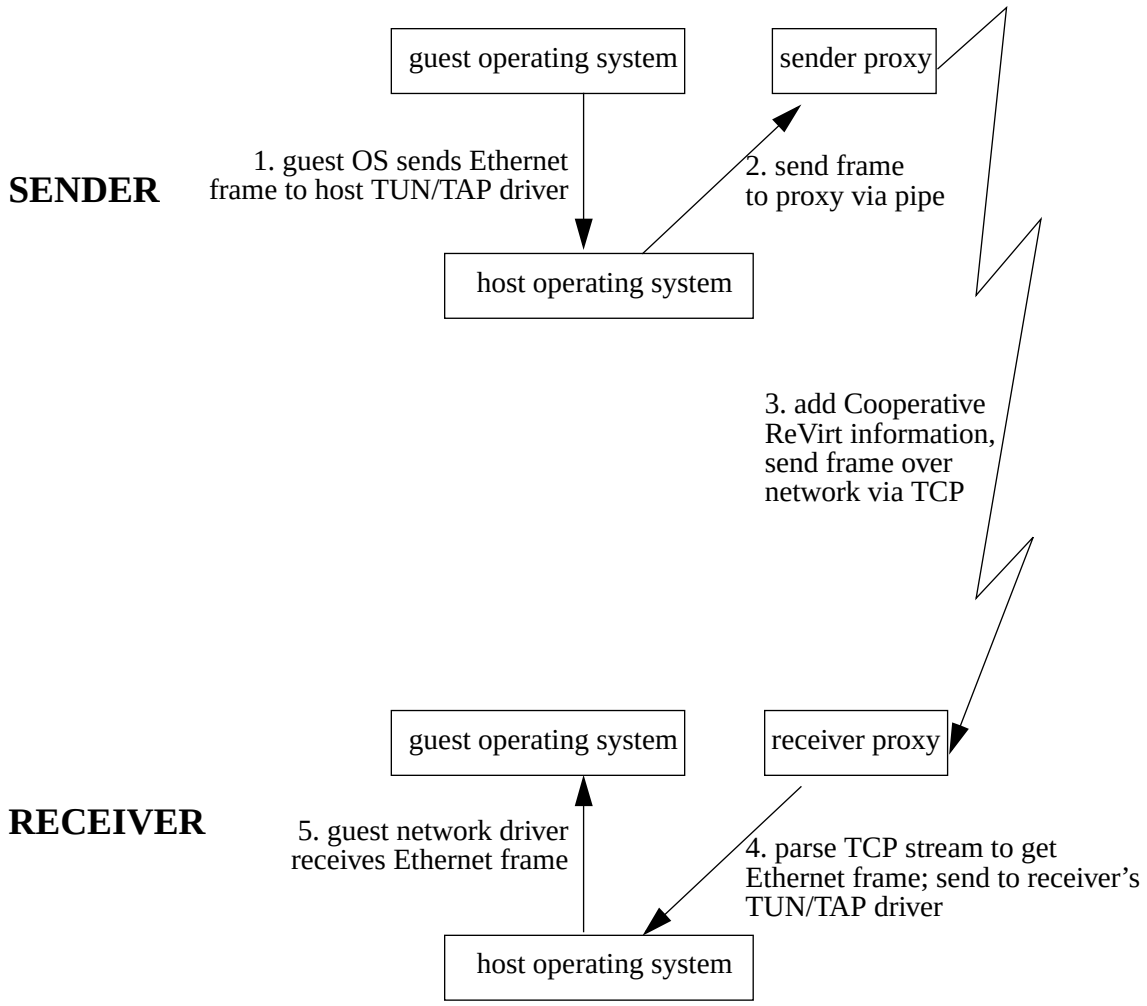


Figure 3: Host-host TCP provides reliable transport for guest Ethernet frames. Cooperative ReVirt provides a reliable communication channel for network packets between virtual machines by forwarding them over a TCP connection between hosts.

may not be willing to rely on the client. Instead, an administrator may want to enable the file server to replay without any help from its clients, especially if the logging layer of the clients is more likely to be compromised than the logging layer of the file server.

These types of asymmetric trust relationships lead to asymmetric (or non-commutative) sets of replaying computers. If computer A is willing to depend on computer B to perform replay, then B will be in A's replay set. At the same time, if computer B is not willing to depend on computer A to perform replay, then A will not be in B's replay set.

Even if the trust relationship between computers is symmetric, the tradeoffs (complexity vs. log savings) of adding another computer to a replay set may dictate asymmetric replay sets. Consider a situation in which computer A sends many messages to computer B, but B sends only a few messages to A. In this situation, B may find it worthwhile to add A to its replay set, but A may not find it worthwhile to add B to its replay set.

Our implementation of Cooperative ReVirt allows an administrator to specify arbitrary replay sets for individual computers. Replay sets may be symmetric or asymmetric depending on trust relationships and expected traffic patterns.

4.5. Dynamic replay sets

A third distinguishing feature of using message logging for intrusion analysis is the generality of applications that must be supported. Message logging has traditionally been used to provide fault tolerance for long-running, scientific computations. In contrast, intrusion analysis focuses on replaying a general set of applications on client and server computers.

One consequence of using message logging for general-purpose workloads is the possibility that traffic patterns may change. Including computer B in computer A's replay set may make sense one day but not another. For example, if B is a local web server, A may use it heavily one day but not another.

Consider how one might want to change computer A's replay set. If A starts receiving a lot of data from computer B, A might choose to add B to its replay set. Adding B to A's replay set immediately allows A to stop logging messages received from B (assuming B is running Cooperative ReVirt, of course). A slightly trickier case is if computer A wants to remove computer B from its replay set, presumably because A wants to not depend on B during replay. Unfortunately, A will still depend on B to regenerate all messages from the beginning of the current checkpoint interval until A started logging B's messages. Hence the benefit of removing B from A's replay set will not materialize until the next coordinated checkpoint is taken.

Cooperative ReVirt supports the ability to dynamically change a computer's replay set. To effect a change to the replay set, Cooperative ReVirt takes a coordinated checkpoint, reads the membership of the new replay set from its configuration file, and continues executing from the checkpoint.

5. Evaluation

Cooperative ReVirt reduces log volume for each computer by avoiding the need to save messages from other computers in that computer's replay set. In this section, we measure this reduction in log volume for a variety of benchmarks. We assess how effectively Cooperative ReVirt would reduce log volume for actual use by measuring the network traffic for several real computers. Finally, we measure the time overhead added by our prototype implementation of Cooperative ReVirt. The experimental setup used for these measurements was a network of computers connected via a 100 Mb/s Ethernet switch. Experiments involving two computers used two AMD Athlon XP 2200+ computers. Experiments involving three computers (SPECweb99) used two Athlon XP 2200+ computers and one 3 GHz Pentium 4 computer.

We measure performance of Cooperative ReVirt on several benchmarks. The first benchmark is compiling the Linux kernel from scratch, where the kernel source tree is stored on a remote NFS server. The second benchmark is PostMark, which was designed to approximate the workload of a file server used for electronic mail, netnews, and web based services [Katcher97]. PostMark creates a large pool of continually changing files. The third benchmark is SPECweb99, which is designed to measure the performance of a web server. For SPECweb99, we use one web server running Apache and two client computers.

Table 2 shows the log volume generated by each benchmark by ReVirt and two configurations of Cooperative ReVirt. Log savings are presented for asymmetric and symmetric replay sets. Cooperative ReVirt is able to reduce log volume by an average of 70% for both clients and servers. Asymmetric configurations of Cooperative ReVirt realize this log savings for computers that are willing to depend on other computers to assist in replay.

We next assess how effectively Cooperative ReVirt would reduce log volume for actual use. We measured the volume of network traffic for three desktop computers, belonging to users in three different research groups in the EECS Department at the University of Michigan. Our first goal was to evaluate how large the replay set for each computer would need to be in order to reduce dramatically the log volume due to network traffic. Our second goal was to evaluate how much of the incoming network traffic was from computers in the same administrative domain (in our case, computers in the eecs.umich.edu domain).

Table 3 shows the results of our measurements. While each of the computers received data from numerous (several hundred to several thousand) sources, the incoming network data was dominated by only a few sources. In fact, for each computer, most of the data received came from a single computer, and

	ReVirt	Symmetric Cooperative Revirt	Asymmetric Cooperative ReVirt (server is in client's replay set; client is not in server's replay set)
NFS kernel compile	client: 2.8 GB/day server: 5.7 GB/day	client: 1.0 GB/day server: 0.9 GB/day	client: 1.0 GB/day server: 5.7 GB/day
PostMark	client: 10.2 GB/day server: 14.3 GB/day	client: 3.0 GB/day server: 3.6 GB/day	client: 3.0 GB/day server: 14.3 GB/day
SPECweb99	client: 13.9 GB/day server: 1.4 GB/day	client: 0.7 GB/day server: 1.0 GB/day	client: 0.7 GB/day server: 1.4 GB/day

Table 2: Log savings from Cooperative ReVirt. Symmetric Cooperative ReVirt reduces the log space required to replay by an average of 70% for the three benchmarks across clients and servers. Asymmetric Cooperative ReVirt yields this log savings for the client because it is willing to depend on the server during replay.

	Fraction of network data received from top N computers		
	N=1	**N=2**	**N=3**
vaniti.eecs.umich.edu	75%	81%	86%
quantify.eecs.umich.edu	91%	94%	95%
tapi.eecs.umich.edu	76%	82%	86%

Table 3: Log savings from Cooperative ReVirt for actual usage. For each computer we measured (vaniti, quanity, tapi), most of the network data received came from a single host, and the top three sources accounted for over 85% of all network data received. In each case, these sources were in the same administrative domain. These results indicate that Cooperative ReVirt would be able to drastically reduce the volume of logged network data in the measured environment.

the top three sources accounted for at least 85% of the total network data received. The top source of data differed for the different computers. For vaniti, the top source was from an NFS-mounted file server that stored the user's home directory. For quantify, the top source was from a computer that stored many of the user's files but not the user's home directory (which was on the local disk). For tapi, the top source was from the computer responsible for backing-up tapi's local disk. The high volume of data received by tapi from the backup server was due to a huge number of small query messages used in the backup protocol (rsync). For each computer we measured, over 85% of incoming network data was from machines in the eecs.umich.edu domain. These results indicate that Cooperative ReVirt would be able to drastically reduce the volume of logged network data by cooperating with a couple machines, all within the same administrative domain.

Finally, we measure the time overhead added by Cooperative ReVirt on the distributed benchmarks (Figure 4). These measurements assume a configuration with symmetric replay sets (all computers are in each other's replay set). Asymmetric replay sets would add less overhead, because some messages would

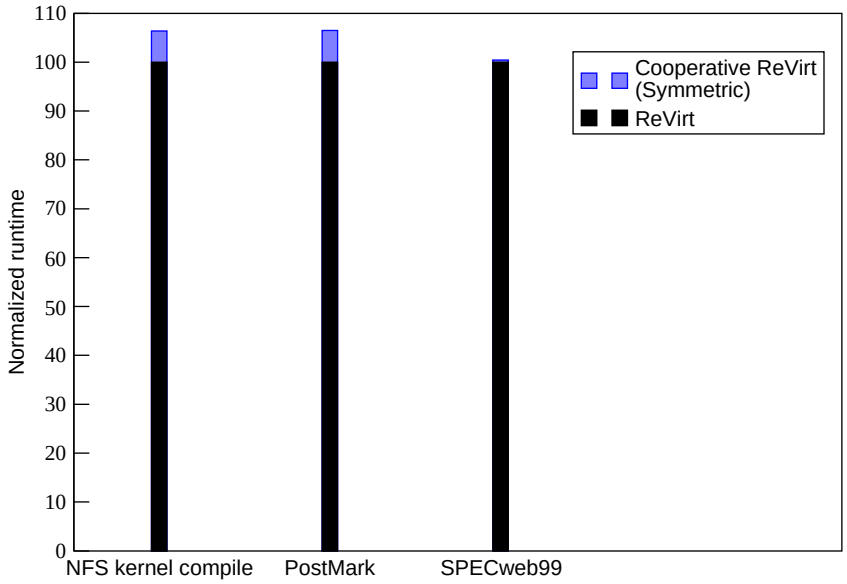


Figure 4: Time overhead of Cooperative ReVirt. Cooperative ReVirt adds 0-6% overhead to ReVirt for the three benchmarks we ran. Sources of overhead are forwarding packets over TCP via a host-host proxy process, computing the SHA-1 hash over the packet header and data, and encrypting the SHA-1 hash with AES.

avoid the processing added by the proxy (adding a secure hash and forwarding over TCP). Cooperative ReVirt adds less than 7% overhead to ReVirt for all benchmarks we ran.

These benchmarks show that Cooperative ReVirt is able to reduce the volume of log data substantially. The additional overhead is reasonable and will drop further if encryption hardware becomes more commonplace.

6. Related work

ReVirt is based on prior work by Bressoud and Schneider on hypervisor-based fault tolerance [Bressoud96]. Bressoud and Schneider use a virtual machine for the PA-RISC architecture to interpose a software layer between the hardware and an unchanged operating system, and they log non-determinism to reconstruct state changes from a primary computer onto its backup. While ReVirt shares several mechanisms with Hypervisor, ReVirt uses them to achieve a different goal. Hypervisor is intended to help tolerate faults by mirroring the state of a primary computer onto a backup. ReVirt takes some of the techniques developed for fault tolerance and applies them to provide intrusion analysis. This difference in goals leads to different design choices. For instance, Hypervisor only seeks to restore the backup to the last saved state of the primary and so discards log records after each synchronization point. In contrast, ReVirt enables replay over long periods (e.g. months) of the computer's execution, so it must save all log records over the period.

Cooperative ReVirt draws on several techniques from the fault-tolerance community, especially from prior work on rollback-recovery [Elnozahy02]. Cooperative ReVirt uses standard rollback-recovery techniques such as coordinated checkpointing and logging of non-deterministic events [Elnozahy94]. Cooperative ReVirt adapts message logging in several ways to make it suitable for the domain of intrusion analysis. The first adaptation is adding a cryptographic hash of the IP header, payload, and sequence number to defend against network attacks during logging or replay. The second adaptation is allowing asymmetric cooperation between hosts to reflect asymmetric trust or traffic patterns. The third adaptation is allowing dynamic membership in each participant's replay set.

7. Conclusions

ReVirt allows one to analyze intrusions in arbitrary detail by replaying the execution of a virtual machine instruction by instruction. The dominant portion of the log data needed to support this replay is incoming network data. Cooperative ReVirt enhances ReVirt with techniques from the message-logging community to reduce the amount of data that must be logged. Using sender-based message logging and coordinated checkpointing, multiple computers running ReVirt can be logged and replayed as a single unit. Cooperative ReVirt adapts message-logging techniques to address the challenges of intrusion analysis. Cooperative ReVirt adds a cryptographic hash of the network header and data to defend against network attacks, allows asymmetric cooperation between hosts to reflect asymmetric trust or traffic patterns, and allows dynamic membership in replay sets to reflect the dynamic nature of general-purpose computing. Cooperative ReVirt is able to reduce the volume of log data by an average of 70% for a variety of distributed computing benchmarks, while adding less than 7% overhead. Measurements of a live network indicate that Cooperative ReVirt would be able to avoid logging 85% of the received network data.

8. References

[Anderson80] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P. Anderson Co., April 1980. Contract 79F296400.

[Barham03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the 2003 Symposium on Operating Systems Principles*, October 2003.

[Bressoud96] Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Transactions on Computer Systems*, 14(1):80–107, February 1996.

[Buchacker01] Kerstin Buchacker and Volkmar Sieh. Framework for testing the fault-tolerance of systems including OS and network aspects. In *Proceedings of the 2001 IEEE Symposium on High Assurance System Engineering (HASE)*, pages 95–105, October 2001.

[Dunlap02] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza Basrai, and Peter M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)*, pages 211–224, December 2002.

[Elnozahy94] E. N. Elnozahy and W. Zwaenepoel. On the Use and Implementation of Message Logging. In *Proceedings of the 1994 International Symposium on Fault-Tolerant Computing (FTCS)*, pages 298–307, June 1994.

[Elnozahy02] E. N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[Goldberg74] Robert P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pages 34–45, June 1974.

[Johnson89] David B. Johnson. Distributed System Fault Tolerance Using Message Logging and Checkpointing. Technical Report COMP TR89-101, Rice University, December 1989. Ph.D. thesis.

[Katcher97] Jeffrey Katcher. PostMark: A New File System Benchmark. Technical Report TR3022, Network Appliance, October 1997.

[King03] Samuel T. King, George W. Dunlap, and Peter M. Chen. Operating System Support for Virtual Machines. In *Proceedings of the 2003 USENIX Technical Conference*, pages 71–84, June 2003.

[Slye98] J. Hamilton Slye and E. N. Elnozahy. Support for Software Interrupts in Log-Based Rollback-Recovery. *IEEE Transactions on Computers*, pages 1113–1123, October 1998.