Tamper Resistant Network Tracing

Andrew G. Miklas[†], Stefan Saroiu[†], Alec Wolman[‡], and Angela Demke Brown[†]
[†]University of Toronto and [‡]Microsoft Research

Abstract: *Raw network traces can be used to compromise the privacy of Internet users. For this reason, many ISPs are reluctant to collect network traces – they often regard possession of such traces as a liability. To mitigate this concern, anonymization techniques have been developed to protect user-identifying information. While most projects anonymize their traces as a post-processing step (i.e., offline), offline anonymization is insecure because raw data may still be exposed during the trace collection and anonymization steps. As an alternative, anonymization can be performed online, drastically reducing the privacy risks associated with storing raw data. Unfortunately, online anonymization is challenging in practice – data must be captured, reconstructed, analyzed, and anonymized at line speed.*

This paper presents a network tracing architecture that combines the performance benefits of offline anonymization with the privacy benefits of online anonymization. Our approach uses a virtual machine and an encrypted file system to protect the raw data allowing it to be securely anonymized offline. In this paper, we present our system's design, and the implementation and evaluation of a simple prototype.

1 Introduction

Modern computer networks and applications are becoming increasingly complex. As a result, understanding the behavior of today's networked systems is harder than ever. Collecting traces of network activity is a powerful and widespread approach to addressing this challenge – it allows network operators to perform traffic engineering and capacity planning; it allows system administrators and application developers to diagnose faults and debug applications; and it allows researchers to examine Internet traffic to help guide the design of future networks and applications.

Despite these benefits, many ISPs avoid collecting network traces because they view possessing them as a liability; raw traces may be used to compromise the privacy of their customers. Protecting user privacy is becoming increasingly important, as evidenced by recent announcements from Web search sites that they will voluntarily limit retention of search logs [20]. For ISPs, the concern about possessing network traces is that they may be revealed, either through leaks or subpoenas. These are not merely hypothetical possibilities: the RIAA has subpoenaed log files while pursuing cases against users suspected of copyright infringement [6]. Oversights or errors in the preparation of traces and logs have also compromised user privacy, resulting in serious consequences [23].

Trace anonymization is the most commonly used technique to address these privacy concerns. This technique uses a one-way transformation to obfuscate any information that can be used to identify a particular user. This allows network researchers and operators to preserve valuable data while simultaneously protecting customer privacy. The most common methodology for anonymization is "offline anonymization," in which the anonymization of the gathered data is done only *after* the entire raw trace has been collected and stored. However, offline anonymization can still lead to exposure of the raw data: until the raw trace is securely deleted, there is the potential for exposure by subpoena or leakage. This potential for privacy loss is becoming more severe as traces contain more sensitive information because of the need to look "deeper" into packets.

Although today most network tracing projects use offline anonymization, current privacy trends make it unlikely that ISPs will continue to accept the risks associated with this approach. Instead, online anonymization can mitigate these risks by obfuscating user-identifying information on the fly. With online anonymization, no raw data is ever stored to disk. Instead, raw packets are reconstructed into higher-level protocols (e.g., TCP, HTTP, SMTP) and relevant information is extracted and anonymized before being written to the disk. Because the raw trace is only stored in volatile memory, online anonymization offers much stronger privacy guarantees than offline anonymization. These additional guarantees can persuade ISPs and network administrators to allow the collection of anonymized network traces. The authors of this paper have experience with tracing Web, P2P, and e-mail traffic at two different Universities. In both cases, anonymizing data online was vitally important to convincing the network operators or the members of Ethics Review committees to authorize our studies [18, 22, 19].

Despite these privacy benefits, there are two challenges that make online anonymization much more difficult to deploy in practice. First, the tracing infrastructure must be sufficiently provisioned to handle the network's peak throughput. In reality, this is very difficult because a network tracing platform must perform resource intensive tasks, such as TCP session reconstruction, application-layer protocol parsing (e.g., HTTP or SOAP), extraction of the relevant data, and anonymization at line speed. As a point of contrast, routers use specialized hardware to perform less resource intensive tasks (e.g., routing and forwarding) at line speed. Second, programmers face significant engineering challenges when developing online analysis and anonymization software. The performance constraints prevent them from using managed language environments, such as Java, Perl, or C#. In addition, unmanaged languages (e.g., C) provide little in the way of memory protection: a bad memory access can crash the tracing system or result in subtle data corruption. Existing libraries (e.g., HTML parsers or regexp engines) also become more difficult to reuse because they often make choices incompatible with the performance needs of an online tracing environment.

In this paper, we present a network tracing architecture that offers the best of both worlds. Our approach combines the operational and software development benefits of offline anonymization with the privacy offered by online anonymization. Our key insight is that offline anonymization can provide the same privacy benefits as online as long as no one can tamper with the trace collection software, the analysis software and the raw trace data. Instead, in our system the trace collection and analysis software is pre-loaded before the raw data is gathered. Once pre-loaded, this software is inaccessible and unmodifiable; users, network operators, and administrators are all unable to interact with the trace processing software. Similarly, the raw data is also inaccessible and unreadable by anything other than the pre-loaded software.

The design of our network tracing system makes the sensitive data (e.g., the raw traces, the encryption keys, and the anonymization keys) inaccessible during trace collection and anonymization. Our architecture further ensures the volatility of all sensitive data, resulting in its destruction upon reboot; only the anonymized trace data is allowed to persist across reboots. By combining the above properties, our system can effectively protect the sensitive data without hardware support such as a Trusted Platform Module (TPM).

Our approach uses a virtual machine (VM) to isolate the tracing environment while the trace data is being gathered and analyzed. The network tracing software that runs in this VM consists of two components. An online component collects the raw trace data, encrypts it with a randomly generated key, and writes it to disk. The encryption key is only stored in volatile memory within the isolated VM – it is never written to disk, and the system ensures that this key is inaccessible outside the isolated VM. The second component performs offline trace analysis and anonymization. This component is also executed inside the isolated virtual machine after the raw trace has been collected, and only the final anonymized trace is written to an exposed disk.

2 Our Design

Our main insight is that a tracing infrastructure can maintain large caches of user-identifiable data without compromising user privacy as long as no user-identifiable data is allowed to leave the host. In this section, we show how a system can be configured to enforce this requirement, achieving the privacy benefits of online anonymization while preserving the attractive usability features of offline anonymization.

2.1 Design Goals

1. Privacy. While the system may store sensitive data such as unanonymized packets, it must never be possible for an outside agent to extract this data.

2. Ease of use. The infrastructure should place as few constraints as possible on the implementation of the analysis software. For example, protocol reconstruction and parsing should not have real-time performance requirements.

3. Robustness. Common bugs found in handling corner cases in the parsing and analysis code should lead to small errors in the trace, rather than crashing the system or completely corrupting its output.

4. Performance. The proposed system must perform as well as today's network tracers when running on equivalent hardware. In particular, it should be possible to trace a high-capacity link with inexpensive hardware.

5. Re-use commodity hardware and software. While the infrastructure could benefit from specialized hardware or software, such as a Trusted Platform Module (TPM), it should be possible to satisfy the above goals using only commodity software and equipment.

2.2 Key Abstractions

Our tracing infrastructure uses two well-known abstractions to protect sensitive data: virtual machines and encryption. In the remainder of this section, we will describe how the combination of these abstractions prevents an adversary from accessing the raw data and the anonymization software, even with full physical access to the hardware.

Virtual machine monitors (VMMs) are often used to provide resource isolation between mutually untrusting VMs [4, 8]. Using virtual machines for isolation is especially beneficial for tasks that require little interaction [2], which is the case with network tracing. We protect all sensitive data associated with collecting and anonymizing network traces. The tracing infrastructure runs all software that processes captured data inside a highly trusted *inaccessible virtual machine* (IVM). As its name implies, users, administrators, and software in other VMs are all unable to interact with the IVM or access any of its internal state once it starts running. To provide this guarantee, we made several modifications to the VMM. We disallowed any untrusted VMs to access DMA-capable devices [7]; we disabled debugging facilities to prevent unauthorized memory accesses; we deactivated standard VMM management functions, such as suspend and migrate; and, finally, we turned off the memory paging mechanisms in the IVM.

Tracing experiments will frequently generate more sensitive data than can fit into memory, imposing a requirement for the safe use of stable storage. For example, a researcher might need to capture a very large raw packet trace before running a multi-pass analysis. VMMs alone cannot protect data written to disk, because the adversary could move the drive to another system and then extract the sensitive data.

We use encryption to ensure that sensitive data stored on the hard disk cannot be retrieved after a reboot. On boot-up, the IVM selects a random key that will be used to encrypt any data written to the hard disk. This key will only be stored in volatile memory assigned to the IVM, ensuring that it is both inaccessible to other VMs and lost on reboot. Because data stored on the disk can only be read with the encryption key, the data is effectively destroyed after a reboot. The use of encryption to make disk storage effectively volatile is not novel; swap file encryption is used on some systems to ensure that fragments of an application's memory space do not persist once the application has terminated or the system restarted [16].

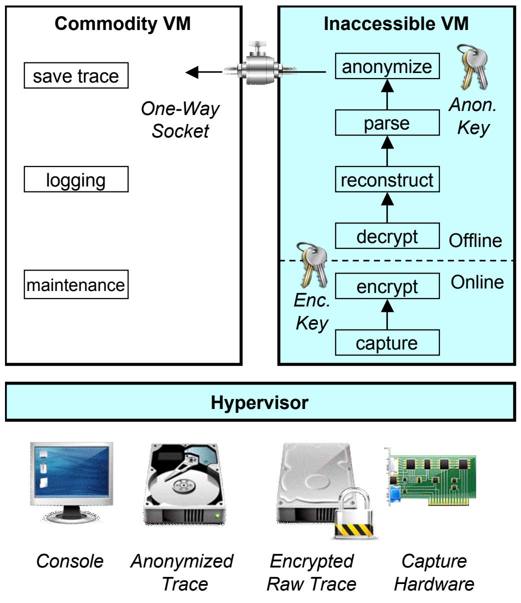

Figure 1. ***The system's overall design.*** *The online component captures packets off the wire, encrypts them, and stores them on disk. All access to sensitive data occurs only within the inaccessible VM. Because the Xen Dom0 domain has full access to the hardware, we use it to contain the IVM.*

2.3 Design Details and Expected Usage

The overall design of our system is shown in Figure 1. At a high level, the system consists of two virtual machines: an inaccessible VM (IVM) that contains all sensitive data and the software to process it, and a commodity VM. The commodity VM allows the researcher to configure the trace system and observe non-sensitive information about the progress of the trace; it is given just enough control over the VMM to start and stop the IVM, and to halt the physical system.

The VMM provides the IVM with access to three I/O devices: the network capture device, a block device, and a one-way socket [24] to the commodity VM. The block device is used to store (encrypted) sensitive data that is too large to keep in memory. The one-way socket allows the IVM to send non-sensitive logging data and the anonymized trace out to the commodity VM. The absence of a bi-directional pipe or virtual network between the VMs adds an additional layer of safety to the system because data never flows from the commodity VM to the IVM. Even if the software running in the IVM has bugs, it is highly unlikely they can be exploited through such a narrow interface.

Prior to beginning a trace, the researcher prepares the disk image used by the IVM. The image will contain a complete operating system, packet capture software, and trace analysis and anonymization software. The prepared system must be able to run from start to finish without any user intervention. When ready to begin the trace, the researcher logs into the commodity VM, copies the disk image to a block device on the system, and instructs the VMM to boot the IVM. At this point, the VMM will unmap the block device containing the trace software from the commodity VM, map the device into the newly instantiated IVM, and boot it.

After booting, the IVM randomly generates the hashing and encryption keys, both of which are considered sensitive data. The hashing key is used to reduce an adversary's ability to conduct a dictionary attack against the anonymized trace. The encryption key is used to encrypt any sensitive data written to the hard drive. We ensure that all sensitive data is encrypted by mounting the IVM's root device read-only, and then mounting an encrypted read-write overlay device. This approach is supported by Linux's logical volume management feature, and has the advantage of automatically encrypting all data written to disk.

Once the system is initialized, packet capture and analysis begin. Conceptually, the packet capture software is responsible for copying all traffic off the network capture device and placing it into a ring buffer backed by encrypted disk storage. Because of the encrypted disk storage, this buffer can support a large backlog of packets. The analysis software reads packets from the buffer and processes them to produce the anonymized trace. The anonymized trace is then sent via the one-way socket to the commodity VM, where it can be viewed by the researcher while the system is running.

The trace software in the IVM also delivers a diagnostic log to the commodity VM using the one-way socket. This log is intended to allow researchers to monitor the progress of the tracer while it is in operation. Information gleaned from the log might assist in debugging or optimizing the tracing software for future runs. Clearly, no sensitive data appears in the diagnostic log.

As part of its configuration, the tracing system is provided with a list of stop conditions that cause the IVM to halt itself automatically. While these conditions can be entirely arbitrary, we expect that typically the system will be configured to stop after the trace meets a specified duration. Once the IVM terminates, the VMM will zero out all memory that was assigned to it. It will then map the block device back into the commodity VM where it can be loaded with new software in preparation for the next tracing run.

2.4 Handling Faults

One serious drawback of most online anonymization techniques is their inability to cope gracefully with bugs in the analysis software. If the tracing software crashes, all data is lost until the system can be restarted. This can result in the loss of megabytes of data, even if the restart process is entirely automated. Worse, this process introduces systematic bias in the data collection, because a long-lived flow is more likely to be affected by a crash than a short-lived flow.

Our system is better able to cope with bugs because the capture and analysis stages are fully decoupled from each other. We can assume that bugs in the capture stage, where a crash would cause data loss, will be very rare for two reasons. First, the capture software is responsible only for capturing packets and loading them into the buffer (encryption can be handled automatically by the file system layer). Because the capture stage has few responsibilities, it will be small and easy to test extensively. Second, there is little reason to change the capture software from one run to another;

nearly all customizations will be implemented in the analysis stage. Because it encompasses much of the system's complexity, most bugs will occur instead in the analysis stage. A crash of this stage does not result in the loss of any network traffic because the capture software can continue to queue up packets while the analysis software is being restarted.

If necessary, our infrastructure allows the analysis software to reconstruct lost flow state by having the restarted analysis software begin reading from the earliest packet still in the buffer, rather than the first unprocessed packet. Of course, care must be taken to remove the packet that triggered the crash prior to restarting the analysis software. We also remove duplicate entries that might result from this "automatic replay" mechanism in a post-processing stage.

2.5 Benefits

Unlike offline anonymization, our approach does not require researchers to work with sensitive data at any time. Because researchers cannot access unanonymized data, they cannot be expected to produce it under a subpoena. Our approach also greatly reduces the chance that unanonymized data will be stolen or accidentally released, because individuals cannot easily extract such data from the system.

The privacy guarantees provided by our tracing system are even stronger than the ones offered by online anonymization. In our system, the hashing key is enclosed inside of an inaccessible virtual machine. While tracing systems that anonymize data online are typically careful not to write unanonymized data out to disk, they generally do not protect the hashing key against theft by an adversary with the ability to log into the machine. In contrast, our approach also prevents anyone from accessing the hashing key.

When encrypted disk space is used to store sensitive data, the analysis code is free to run offline at much slower than line speeds, because the disk can be used to store the raw packets for later processing. In extreme cases, the entire analysis can be deferred until after the raw trace is gathered. The flexible performance requirements imposed by our system allow the researcher to use managed languages and sophisticated libraries when creating the analysis software. As a result, the code is both easier to write and less likely to contain bugs.

While advanced languages and libraries can help the researcher to produce correct code, it is unrealistic to expect that complex protocol parsers will be entirely bug-free. Our tracing infrastructure allows the analysis software to fail gracefully in the presence of such bugs, as described in Section 2.4. In contrast, when online parsers crash, they generally lose data from all flows until they are restarted.

Lastly, our system is built using only off-the-shelf hardware and software, which provides many practical benefits. For example, the entire system can be run on an ordinary and inexpensive PC. Software components, such as the Xen VMM, can be freely downloaded from the Web. Furthermore, our system can take advantage of performance enhancements to VMM software as they become available.

3 Security Analysis

In this section, we start by describing the security assumptions we made. Next, we identify the threats to user privacy introduced by our network tracing infrastructure and characterize its ability to deal with those threats.

3.1 Our Assumptions

Our design is centered on four assumptions: (1) the VMM prevents software in one VM from reading memory assigned to another; (2) memory is cleared on a reboot; (3) it is impractical to physically extract data from a typical PC without triggering a reboot; (4) it is infeasible to decrypt data without the key.

We use VMMs to protect all sensitive data such as the encryption and anonymization keys. Virtual machine monitors are trusted to maintain isolation between VMs in many mission critical and security driven applications [4, 8]. While both VMMs and OS kernels are responsible for isolating units of software from each other, VMMs are generally thought to provide stronger security guarantees because they are small enough to be rigorously verified and export only a very narrow interface to VMs. In contrast, OS kernels tend to be very large and complex pieces of software that supply very rich interfaces to their clients.

Memory that retains its value across reboots [3] may compromise the security of our system. For example, an adversary could insert a boot-CD containing an ordinary Linux installation and then physically reboot the tracing system. He could then dump out physical memory in an attempt to locate the decryption key. Because the VMM is no longer in control of the system, it cannot protect the sensitive data from capture. This vulnerability can be removed if the BIOS performs a full memory test before transferring control to the boot media, which generally involves initializing every address to some known value. Of course, care would have to be taken to ensure that the user cannot abort or otherwise disable the memory test.

While VMMs protect the sensitive data from logical attack, they cannot protect the data from an attacker with physical access. Fortunately, it is not straightforward to physically extract data from the memory of an ordinary PC. An attacker would need to physically attach hardware such as a bus analyzer to the tracing host without triggering a reboot. To protect against such attacks, additional physical security measures must be employed [21] such as coating the system's mainboard with epoxy [1].

3.2 Threat Model

The first threat we consider is that of subpoena. ISPs are discovering that logs and traces gathered for diagnostic and research purposes can be used in court proceedings against their customers. A detailed trace of P2P activity that may be very useful to the research community [9, 15, 10] would also be of great benefit to the RIAA: it would allow them to identify users who are inappropriately sharing copyrighted content. If the RIAA subpoenas a trace, the ISP will be required by law to turn it over, even though the ISP has no business incentive to co-operate with the RIAA and may in fact have good reasons not to. As a result, ISPs may view the benefits of collecting network traces as being outweighed by the liability of possessing information that could be subpoenaed and then used to compromise the privacy of their customers.

Our tracing architecture was largely motivated by the need to conduct traces while still preserving user privacy, even if a court requires that someone be allowed full access to any existing traces and to the trace infrastructure. Our system can protect user privacy even if the adversary has physical access to the tracing machine and has full administrative privileges to login to the machine. In other words, once a tracing experiment has been initiated, the sensitive information such as the raw trace and the anonymization key are protected from the system administrator in the same way that they are protected from any adversary. Therefore, our solution makes it technically infeasible not only to turn over the hashing key used in the anonymization process, but also to turn over any raw traces that have not yet been anonymized. Being forced to turn over the tracing system simply leads to a loss of any tracing data that was collected but had not yet been anonymized.

Another potential threat to a network tracing system is remote theft of the anonymized data collected by the tracing machine. There are many possible ways to break into a system over the network, yet there is one simple solution that eliminates this entire class of attacks. We configure the port on the network switch to which the tracing machine is attached so that it can only be used for port mirroring – any traffic transmitted by the tracing machine will be discarded by the network switch. This effectively prevents any trace data from being transmitted over the network by an unauthorized user (or any user, for that matter).

3.3 Unaddressed Attacks

Our tracing architecture does not protect against faulty anonymization policies. We have no means to ensure that the data written to the anonymized trace is in fact fully anonymous. Unfortunately, there is no simple checklist or set of rules that can be followed to ensure that a trace does not leak private data. In fact, even a rigorous manual audit of the trace anonymization software can miss certain properties and anomalies that could be exploited by a determined adversary [12]. Fortunately, there are tools that can aid in the design and implementation of sound anonymization policies [13].

Our tracing system is also susceptible to traffic injection attacks. To perform a packet injection attack, an adversary transmits traffic on the network being traced, then later identifies this traffic in the anonymized trace. For example, suppose the adversary knows that an anonymized trace of HTTP traffic is being collected and wants to learn how many other users on the network are visiting a certain Web site. The adversary generates HTTP GET requests to the Web site of interest at known times and with an unusual size (most GET requests are small). The adversary then learns the anonymized name of the Web site of interest by scanning through the anonymized trace searching for requests that match the generated times and sizes.

Packet injection attacks do not completely break the anonymization: for example, they do not allow the adversary to deduce the anonymization key. However, they may allow the adversary to derive sensitive data from the anonymized trace. The best way to defend against such attacks is simply to avoid public release of the anonymized trace data.

3.4 Operational Attacks

In addition to threats to user privacy, network tracers are also subject to another class of incidental attacks. Any attack that traverses the network link being monitored by the network tracing system may also incidentally affect the tracing system itself. This is especially a problem when tracing networks directly connected to the Internet because hosts routinely receive attack traffic such as vulnerability probes, denial-of-service (DoS) attacks, and backscatter from attacks occurring elsewhere on the Internet [14]. Methods exist to reduce the impact of DoS attacks [11] and adversarial traffic [5]; however, these methods may have limited effectiveness against a large enough attack. Tracers that use offline anonymization are more resilient to such attacks because they need not process the attack traffic in real time.

4 Implementation and Evaluation

In this section, we describe an implemented prototype of the online portion of our system. Our goal is to evaluate the performance implications of running a packet capture program encapsulated in a virtual machine and recording a raw trace to an encrypted file system. Because one of our design goals is to reuse commodity software and hardware, we used off-the-shelf software and hardware components in all our experiments. We believe that our prototype's performance can be easily increased by better provisioning the hardware and optimizing the software.

Our tracing host uses *tcpdump* version 3.9.5 to collect the packet traces. For virtual machine isolation, we use the Xen 3.0.3-1 hypervisor. We use the Xen Dom0 privileged domain to host the inaccessible virtual machine (IVM) giving the tracing software direct access to the machine's physical network card and hard disk. In this way, we reduce the overhead introduced by virtualization.

Our use of the Xen infrastructure is unusual. In typical Xen-based systems, most software runs outside Dom0 to protect privileged operations from buggy or malicious code. Instead, our system runs the bulk of the functionality in the privileged domain. We made this decision for a number of reasons. First, because Dom0 is privileged it must be inaccessible. Also, because the capture software and Dom0 are both inaccessible, they can both run within the same isolation boundary. Finally, the capture software has fast access to the hardware when running in Dom0.

Our tracing host is a dual Intel Xeon 3.6GHz (four cores) with 1 GB of RAM, a Fujitsu MAP3367MP SCSI disk, and an Intel 82541 gigabit NIC. We are running Linux Debian 4.0 (etch), kernel version 2.6.18-4. This machine is attached to a dedicated Dell PowerConnect 2724 gigabit switch to which two other commodity PCs are also attached. One PC sends constant bit-rate (CBR) traffic at a configurable rate to the other PC, while the switch is configured to mirror all traffic and send it to our tracing host. The two PCs are configured to send traffic at a rate of up to 700 Mbps. In addition, we verified that no packets were being lost by the switch.

We used the *dm-crypt* [17] device-mapper module from the Linux 2.6 kernel to encrypt the captured data. This module provides a simple abstraction – it adds an encrypted device on top of any ordinary block device without the need for any support from a file system or an application. The dm-crypt module supports several encryption schemes; the one we used was a carefully optimized implementation of AES.

We perform a brief evaluation of our prototype. The goal of our evaluation is to quantify the performance overhead introduced by the virtualization layer and by the encryption module. For this, we measure the packet loss rate of our tracing host when capturing UDP traffic sent at a constant rate (CBR) when the rate varies from 30,000 to 60,000 packets per second. Because each packet is 1500 bytes, this corresponds to 340 to 690 Mbps.

Figure 2 shows the loss rate experienced by our host under four configurations. First, we evaluated "tcpdump" running without encryption and without a VM. We found that once the packet rate reached 486 Mbps, the system started to experience loss (less than 0.3%). Because our host has a single disk, the disk's write speed becomes the system's bottleneck at high packet rates. In a separate experiment, we found that our SCSI disk has a maximum write rate of about 60 Mbytes per second (about 480 Mbps).

Second, when we ran "tcpdump" inside the Xen Dom0 privileged domain, we found that the virtualization layer added an insignificant overhead – at a packet rate of 486 Mbps, the loss rate was less than 0.9%. Third, when we ran "tcpdump" with encryption but without a VM, we also found that the loss rate was about 0.3% at a packet rate of 486 Mbps. Finally, we evaluated our prototype with both encryption and virtualization. This configuration experienced worse performance – at a packet rate of 486 Mbps, the loss rate was 8%. However, the system experienced no losses once the packet rate was reduced by 5% to 460 Mbps. We found this preliminary experiment very encouraging. Even with off-the-shelf hardware and software components, the overhead of virtualization and encryption is small, making our tracing platform practical.

5 Summary

This paper presented a network tracing architecture that combines the performance benefits of offline anonymization with the privacy offered by online anonymization. We used virtual machines technology and encryption to protect the raw data allowing it to be securely anonymized offline. We also described an implementation and an evaluation of a simple prototype.

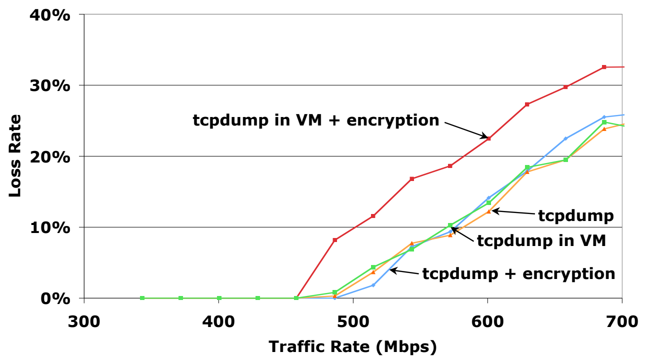

Figure 2. ***Evaluation of a prototype of our system.*** *We measured the loss rate as a function of the traffic rate. As traffic increases past 460 to 480Mbps, our systems starts to experience losses. This rate corresponds to the maximum write speed of our disk. The performance overhead incurred by virtualization and encryption is small.*

Acknowledgments

We would like to thank Gianluca Iannaccone and Intel Research for the generous support with the hardware needs of this project. We are grateful to Yashar Ganjali, Andy Warfield, and the anonymous reviewers for their feedback.

References

[1] R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. In *Proc. of the 2nd USENIX Workshop on Electronic Commerce*, Oakland, CA, November 1996.

[2] S. M. Bellovin. Virtual machines, virtual security. *Communications of the ACM*, 49(10), 2006.

[3] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum. Shredding your garbage: reducing data lifetime through secure deallocation. In *Proc. of the 14th USENIX Security Symposium*, Baltimore, MD, July 2005.

[4] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *Proc. of the 27th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.

[5] S. Dharmapurikar and V. Paxson. Robust TCP stream reassembly in the presence of adversaries. In *Proc. of the 14th USENIX Security Symposium*, Baltimore, MD, July 2005.

[6] Electronic Frontier Foundation. RIAA v. Verizon case archive. `http://www.eff.org/legal/cases/RIAA_v_Verizon/`.

[7] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *Proc. of the 1st ACM workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure (OASIS)*, Boston, MA, October 2004.

[8] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proc. of the 19th ACM symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, October 2003.

[9] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of the 19th*

ACM symposium on Operating Systems Principles (SOSP), Bolton Landing, NY, October 2003.

[10] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *Proc. of the 5th ACM Internet Measurement Conference (IMC)*, Berkeley, CA, October 2005.

[11] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2):39–53, 2004.

[12] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, 2006.

[13] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. of the 9th ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003.

[14] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proc. of the 4th ACM Internet Measurement Conference (IMC)*, Taormina, Italy, October 2004.

[15] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, February 2005.

[16] N. Provos. Encrypting virtual memory. In *Proc. of the 9th USENIX Security Symposium*, Denver, CO, August 2000.

[17] C. Saout. dm-crypt: a device-mapper crypto target, 2007. `http://www.saout.de/misc/dm-crypt/`.

[18] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of Internet content delivery systems. In *Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[19] S. Saroiu and T. Ronda. Using network tracing to address internet phishing, 2006. Project proposal to the Ethics Review Committee at the University of Toronto – `http://www.slup.cs.toronto.edu/utmtrace/ethics/ethics.html`.

[20] B. Stone. The most privacy-friendly search engine on the web is... *New York Times*, July 2007.

[21] S. H. Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. *Lecture Notes in Computer Science*, 1965:302–317, 2000.

[22] A. Wolman. *Sharing and Caching Characteristics of Internet Content*. PhD thesis, University of Washington, Seattle, WA, 2002.

[23] T. Zeller Jr. AOL executive quits after posting of search data. *International Herald Tribune*, August 2006.

[24] X. Zhang, S. McIntosh, P. Rohatgi, and J. L. Griffin. XenSocket: A high-throughput interdomain transport for VMs. Technical Report RC24247, IBM Research, 2007.