# Bunker: A Privacy-Oriented Platform for Network Tracing

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

***Abstract:*** *ISPs are increasingly reluctant to collect and store raw network traces because they can be used to compromise their customers' privacy. Anonymization techniques mitigate this concern by protecting sensitive information. Trace anonymization can be performed* offline *(at a later time) or* online *(at collection time). Offline anonymization suffers from privacy problems because raw traces must be stored on disk – until the traces are deleted, there is the potential for accidental leaks or exposure by subpoenas. Online anonymization drastically reduces privacy risks but complicates software engineering efforts because trace processing and anonymization must be performed at line speed. This paper presents Bunker, a network tracing system that combines the software development benefits of offline anonymization with the privacy benefits of online anonymization. Bunker uses virtualization, encryption, and restricted I/O interfaces to protect the raw network traces and the tracing software, exporting only an anonymized trace. We present the design and implementation of Bunker, evaluate its security properties, and show its ease of use for developing a complex network tracing application.*

## 1 Introduction

Network tracing is an indispensable tool for many network management tasks. Operators need network traces to perform routine network management operations, such as traffic engineering [19], capacity planning [38], and customer accounting [15]. Several research projects have proposed using traces for even more sophisticated network management tasks, such as diagnosing faults and anomalies [27], recovering from security attacks [45], or identifying unwanted traffic [9]. Tracing is also vital to networking researchers. As networks and applications grow increasingly complex, understanding the behavior of such systems is harder than ever. Gathering network traces helps researchers guide the design of future networks and applications [42, 49].

Customer privacy is a paramount concern for all online businesses, including ISPs, search engines, and e-commerce sites. Many ISPs view possessing raw network traces as a liability: such traces sometimes end up compromising their customers' privacy through leaks or subpoenas. These concerns are real: the RIAA has subpoenaed ISPs to reveal customer identities when pursuing cases of copyright infringement [16]. Privacy concerns go beyond subpoenas, however. Oversights or errors in preparing and managing network trace and server log files can seriously compromise users' privacy by disclosing social security numbers, names, addresses, or telephone numbers [5, 54].

Trace anonymization is the most common technique for addressing these privacy concerns. A typical implementation uses a keyed one-way secure hash function to obfuscate sensitive information contained in the trace. This could be as simple as transforming a few fields in the IP headers, or as complex as performing TCP connection reconstruction and then obfuscating data (e.g., email addresses) deep within the payload. There are two current approaches to anonymizing network traces: *offline* and *online*. Offline anonymization collects and stores the entire raw trace and then performs anonymization as a post-processing step. Online anoymization is done on-the-fly by extracting and anonymizing sensitive information before it ever reaches the disk. In practice, both methods have serious shortcomings that make network trace collection increasingly difficult for network operators and researchers.

Offline anonymization poses risks to customer privacy because of how raw network traces are stored. These risks are growing more severe because of the need to look "deeper" into packet payloads, revealing more sensitive information. Current privacy trends make it unlikely that ISPs will continue to accept the risks associated with offline anonymization. We have first-hand experience with tracing Web, P2P, and e-mail traffic at two universities. In both cases the universities deemed the privacy risks associated with offline anonymization to be unacceptable.

While online anonymization offers much stronger privacy benefits, it is very difficult to deploy in practice because it creates significant software engineering issues. Any portion of the trace analysis that requires access to sensitive data must be performed on-the-fly and at a rate that can handle the network's peak throughput. This is practical for simple tracing applications that analyze only IP and TCP headers; however, it is much more difficult for tracing applications that require deep packet inspection. Developing complex online tracing software therefore poses a significant challenge. Developers are limited in their selection of software: adopting garbage-collected (e.g., Java, C#) and dynamic scripting (e.g., Python, Perl) languages can be difficult; reusing existing libraries (e.g., HTML parsers or regexp engines) may also be hard if their implementation choices are incompatible with performance requirements. A network tracing experiment illustrates the performance challenges of online tracing.

Our goal was to run hundreds of regular expressions to identify phishing Web forms. However, an Intel 3.6GHz processor running just one of these regular expressions (using the off-the-shelf "libpcre" regexp library) could only handle less than 50 Mbps of incoming traffic.

This paper presents Bunker, a network tracing system built and deployed at the University of Toronto. Bunker offers the software development benefits of offline anonymization and the privacy benefits of online anonymization. Our key insight is that we can use the buffer-on-disk approach of offline anonymization if we can "lock down" the trace files and trace analysis software. This approach lets Bunker avoid all the software engineering downsides of online trace analysis. To implement Bunker, we use virtual machines, encryption, and restriction of I/O device configuration to construct a *closed-box* environment; Bunker requires no specialized hardware (e.g., a Trusted Platform Module (TPM) or a secure co-processor) to provide its security guarantees. The trace analysis and anonymization software is preloaded into a closed-box VM before any raw trace data is gathered. Bunker makes it difficult for network operators to interact with the tracing system or to access its internal state once it starts running and thereby protects the anonymization key, the tracing software, and the raw network trace files inside the closed-box environment. The closed-box environment produces an anonymized trace as its only output.

To protect against physical attacks (e.g., hardware tampering), we design Bunker to be *safe-on-reboot*: upon a reboot, all sensitive data gathered by the system is effectively destroyed. This property makes physical attacks more difficult because the attacker must tamper with Bunker's hardware without causing a reboot. While a small class of physical attacks remains feasible (e.g., cold boot attacks [21]), in our experience ISPs find the privacy benefits offered by a closed-box environment that is safe-on-reboot a significant step forward. Although the system cannot stop ISPs from being subject to wiretaps, Bunker helps protect ISPs against the privacy risks inherent in collecting and storing network traces.

Bunker's privacy properties come at a cost. Bunker requires the network operator to pre-plan what data to collect and how to anonymize it before starting to trace the network. Bunker prevents anyone from changing the configuration while tracing; it can be reconfigured only through a reboot that will erase all sensitive data.

The remainder of this paper describes Bunker's threat model (Section 2), design goals and architecture (Section 3), as well as the benefits of Bunker's architecture (Section 4). It then analyzes Bunker's security properties when confronted with a variety of attacks (Section 5), describes operational issues (Section 6), and evaluates Bunker's software engineering benefits by examining a tracing application (phishing analysis) built by one student in two months that leverages off-the-shelf components and scripting languages (Section 7). The paper's final sections review legal issues posed by Bunker's architecture (Section 8) and related work (Section 9).

## 2 Threat Model

This section outlines the threat model for network tracing systems. We present five classes of attacks and discuss how Bunker addresses each.

### 2.1 Subpoenas For Network Traces

ISPs are discovering that traces gathered for diagnostic and research purposes can be used in court proceedings against their customers. As a result, they may view the benefits of collecting network traces as being outweighed by the liability of possessing such information. Once a subpoena has been issued, an ISP must cooperate and reveal the requested information (e.g., traces or encryption keys) as long as the cooperation does not pose an undue burden. Consequently, *a raw trace is protected against a subpoena only if no one has access to it or to the encryption and anonymization keys used to protect it.*

Our architecture was designed to collect traces while preserving user privacy even if a court permits a third party to have full access to the system. Once a Bunker trace has been initiated, all sensitive information is protected from the system administrator in the same way it is protected from any adversary. Thus, our solution makes it a hardship for the ISP to surrender sensitive information. We eliminate potential downsides to collecting traces for legitimate purposes but do not prevent those with legal wiretap authorization from installing their own trace collection system.

### 2.2 Accidental Disclosure

ISPs face another risk, that of accidental disclosure of sensitive information from a network trace. History has shown that whenever people handle sensitive data, the danger of accidental disclosure is substantial. For example, the British Prime Minister recently had to publicly apologize when a government agency accidentally lost 25 million child benefit records containing names and bank details because the agency did not follow the correct procedure for sending these records by courier [5]. Bunker vastly reduces the risk that sensitive data will be accidentally released or stolen because no human can access the unanonymized trace.

### 2.3 Remote Attacks Over The Internet

Remote theft of data collected by a tracing machine presents another threat to network tracing systems. 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. To collect traces, Bunker uses a specialized network capture card that is incapable of sending outgoing data. It also uses firewall rules to limit access to the tracing machine from the internal private network. Section 5.3 examines in-depth Bunker's security measures against such attacks.

### 2.4 Operational Attacks

Attacks that traverse the network link being monitored, such as denial-of-service (DoS) attacks, may also incidentally affect the tracing system. This is a problem when tracing networks with direct connections to the Internet: Internet hosts routinely receive attack traffic such as vulnerability probes, denial-of-service (DoS) attacks, and back-scatter from attacks occurring elsewhere on the Internet [36]. Methods exist to reduce the impact of DoS attacks [31] and adversarial traffic [13]. However, these methods may have limited effectiveness against a large enough attack. Both Bunker and offline anonymization systems are more resilient to such attacks because they need not process the traffic in real time.

Because many network studies collect traces for relatively long time periods, an attacker with physical access could tamper with the monitoring system after it has started tracing, creating the appearance that the original system is still running. For example, the attacker might reboot the system and then set up a new closed-box environment that uses anonymization keys known to the attacker. Section 6 describes a simple modification to Bunker that addresses this type of attack.

### 2.5 Attacks On Anonymization

Packet injection attacks attempt to partially learn the anonymization mapping by injecting traffic and then analyzing the anonymized trace. To perform such attacks, an adversary transmits traffic over the network being traced and later identifies this traffic in the anonymized trace. These attacks are possible when non-sensitive trace information (e.g., times or request sizes) is used to correlate entries in the anonymized trace with the specific traffic being generated by the adversary. Packet injection attacks do not completely break the anonymization mapping because they do not let the adversary deduce the anonymization key. Even without packet injection, recent work has shown that private information can still be recovered from data anonymized with state-of-the-art techniques [10, 34]. These attacks typically make use of public information and attempt to correlate it with the obfuscated data. Our tracing system is susceptible to attacks on the anonymization scheme. The best way to defend against this class of attacks is to avoid public release of anonymized trace data [10].

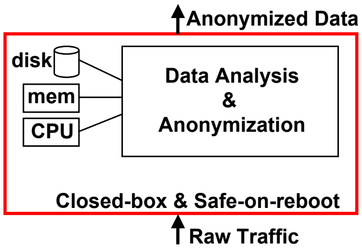

Figure 1. ***Logical view of Bunker:*** *Raw data enters the closed-box perimeter and only anonymized data leaves this perimeter.*

Another problem involves ensuring that the anonymization policy is specified correctly, and that the implementation correctly implements the specification. Bunker does not explicitly address these issues. We recommend code reviews of the trace analysis and anonymization software. However, even a manual audit of this software can miss certain properties and anomalies that could be exploited by a determined adversary [34]. Although there is no simple checklist to follow that ensures a trace does not leak private data, there are tools that can aid in the design and implementation of sound anonymization policies [35].

### 2.6 Summary

Bunker's design raises the bar for mounting any of these attacks successfully. At a high level, our threat model assumes that: (1) the attacker has physical access to the tracing infrastructure but no specialized hardware, such as a bus monitoring tool; (2) the attacker did not participate in implementing the trace analysis software. While Bunker's security design is motivated by the threat of subpoenas, it also addresses the other four classes of attacks described in this section. We examine security attacks against Bunker in Section 5 and we discuss legal issues in Section 8.

## 3 The Bunker Architecture

Our main insight when designing Bunker is that a tracing infrastructure can maintain large caches of sensitive data without compromising user privacy as long as none of that data leaves the host. Figure 1 illustrates Bunker's high-level design, which takes raw traffic as input and generates an anonymized trace.

### 3.1 Design Goals

**1. Privacy.** While the system may store sensitive data such as unanonymized packets, it must not permit an outside agent to extract anything other than analysis output.

**2. Ease of development.** The system should place as few constraints as possible on implementing 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 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. Use commodity hardware and software.** The proposed system should not require specialized hardware, such as a Trusted Platform Module (TPM).

## 3.2 Privacy Properties

To meet our privacy design goal, we must protect all gathered trace data even from an attacker who has physical access to the network tracing platform. To achieve this high-level of protection, we designed Bunker to have the following two properties:

**1. Closed-box.** The tracing infrastructure runs all software that has direct access to the captured trace data inside a closed-box environment. Administrators, operators, and users cannot interact with the tracing system or access its internal state once it starts running. Input to the closed-box environment is raw traffic; output is an anonymized trace.

**2. Safe-on-reboot.** Upon a reboot, all gathered sensitive data is effectively destroyed. This means that all unencrypted data is actually destroyed; the encryption key is destroyed for all encrypted data placed in stable storage. Bunker uses ECC RAM modules that are zeroed out by the BIOS before booting [21]. Thus, it is safe-on-reboot for reboots caused by pressing the RESET button or by powering off the machine.

The closed-box property prevents an attacker from gaining access to the data or to the tracing code while it is running. However, this property is not sufficient. An attacker could restart the system and boot a different software image to access data stored on the tracing system, or an attacker could tamper with the tracing hardware (e.g., remove a hard drive and plug it in to another system). To protect sensitive data against such physical attacks, we use the safe-on-reboot property to erase all sensitive data upon a reboot. Together, these two properties prevent an attacker from gaining access to sensitive data via system tampering.

## 3.3 The Closed-Box Property

Bunker uses virtual machines to provide the closed-box property. We now describe the rationale for our design and implementation.

### 3.3.1 Design Approach

In debating whether to use virtual or physical machines (e.g., a sealed appliance) to design our closed-box environment, we chose the virtual machine option primarily for flexibility and ease of development. We anticipated that our design would undergo small modifications to accommodate unforeseen problems and worried that making small changes to a sealed appliance would be too difficult after the initial system was implemented and deployed. With VMs, Bunker's software can be easily retrofitted to trace different types of traffic. For example, we used Bunker to gather a trace of Hotmail e-mails and to gather flow-level statistics about TCP traffic.

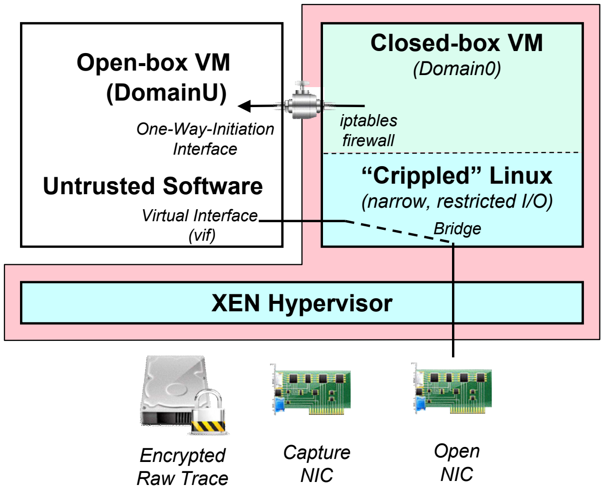

Figure 2. ***Overview of Bunker's implementation.*** *The closed-box VM runs a carefully configured Linux kernel. The shaded area represents the Trusted Computing Base (TCB) of our system.*

Virtual machine monitors (VMMs) have been used in the past for building closed-box VMs [20, 11]. Using virtual machines to provide isolation is especially beneficial for tasks that require little interaction [6], such as network tracing. Bunker runs all software that processes captured data inside a highly trusted closed-box VM. Users, administrators, and software in other VMs cannot interact with the closed-box or access any of its internal state once it starts running.

### 3.3.2 Implementation Details

We used the Xen 3.1 VMM to implement Bunker's closed-box environment. Xen, an open-source VMM, provides para-virtualized x86 virtual machines [4]. The VMM executes at the highest privilege level on the processor. Above the VMM are the virtual machines, which Xen calls *domains*. Each domain executes a guest operating system, such as Linux, which runs at a lower privilege level than the VMM.

In Xen, Domain0 has a special role: it uses a control interface provided by the VMM to perform management functions outside of the VMM, such as creating other domains and providing access to physical devices (including the network interfaces). Both its online trace

```
iptables -P INPUT DROP
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state NEW,ESTABLISHED -j ACCEPT
```

Figure 3. ***iptables firewall rules:*** *An abbreviated list of the rules that creates a one-way-initiation interface between the closed-box VM and the open-box VM. These rules allow connections only if they are initiated by the closed-box VM. Note that the ESTABLISHED state above refers to a connection state used by iptables and not to the ESTABLISHED state in the TCP stack.*

collection and offline trace analysis components are implemented as a collection of processes that execute on a "crippled" Linux kernel that runs in the Domain0 VM, as shown in Figure 2.

We carefully configured the Linux kernel running in Domain0 to run as a closed-box VM. To do this, we severely limited the closed-box VM's I/O capabilities and disabled all the kernel functionality (i.e., kernel subsystems and modules) not needed to support tracing. We disabled all drivers (including the monitor, mouse and keyboard) inside the kernel except for: 1) the network capture card driver; 2) the hard disk driver; 3) the virtual interface driver, used for closed-box VM to open-box VM communication, and 4) the standard NIC driver used to enable networking in the open-box VM. We also disabled the login functionality; nobody, ourselves included, can login to the closed-box VM. Once the kernel boots, the kernel init process runs a script that launches the tracer. We provide a publicly downloadable copy of the kernel configuration file[1] used to compile the Domain0 kernel so that anyone can audit it.

The closed-box VM sends anonymized data and non-sensitive diagnostic data to the open-box VM via a *one-way-initiation interface*, as follows. We setup a layer-3 firewall (e.g., iptables) that allows only those connections initiated by the closed-box VM; this firewall drops any unsolicited traffic from the open-box VM. Figure 3 presents an abbreviated list of the firewall rules used to configure this interface.

We deliberately *crippled* the kernel to restrict all other I/O except that from the four remaining drivers. We configured and examined each driver to eliminate any possibility of an adversary taking advantage of these channels to attack Bunker. Section 5 describes Bunker's system security in greater detail.

## 3.4 The Safe-on-Reboot Property

To implement the safe-on-reboot property, we need to ensure that all sensitive data and the anonymization key are stored in volatile memory only. However, tracing experiments frequently generate more sensitive data than can fit into memory. For example, a researcher might need to capture a very large raw packet trace before running a trace analysis program that makes multiple passes through the trace. VMMs alone cannot protect data written to disk, because an adversary could simply move the drive to another system to extract the data.

### 3.4.1 Design Approach

On boot-up, the closed-box VM selects a random key that will be used to encrypt any data written to the hard disk. This key (along with the anonymization key) is stored only in the closed box VM's volatile memory, ensuring that it is both inaccessible to other VMs and lost on reboot. Because data stored on the disk can be read only with the encryption key, this approach effectively destroys the data 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 has restarted [39].

### 3.4.2 Implementation Details

To implement the safe-on-reboot property, we need to ensure that all sensitive information is either stored only in volatile memory or on disk using encryption where the encryption key is stored only in volatile memory. To implement the encrypted store, we use the *dm-crypt* [41] device-mapper module from the Linux 2.6.18 kernel. This module provides a simple abstraction: it adds an encrypted device on top of any ordinary block device. As a result, it works with any file system. The dm-crypt module supports several encryption schemes; we used the optimized implementation of AES. To ensure that data in RAM does not accidentally end up on disk, we disabled the swap partition. If swapping is needed in the future, we could enable dm-crypt on the swap partition. The root file system partition that contains the closed-box operating system is initially mounted read only. Because most Linux configurations expect the root partition to be writable, we enable a read-write overlay for the root partition that is protected by dm-crypt. This also ensures that the trace analysis software does not accidentally write any sensitive data to disk without encryption.

## 3.5 Trace Analysis Architecture

Bunker's tracing software consists of two major pieces: 1) the online component, independent of the particular network tracing experiment, and 2) the offline component, which in our case is a phishing analysis tracing application. Figure 4 shows Bunker's entire pipeline, including the online and offline components.

Bunker uses *tcpdump* version 3.9.5 to collect packet traces. We fine-tuned tcpdump to increase the size of its

[1] `http://www.slup.cs.toronto.edu/utmtrace/config-2.6.18-xen0-noscreen`

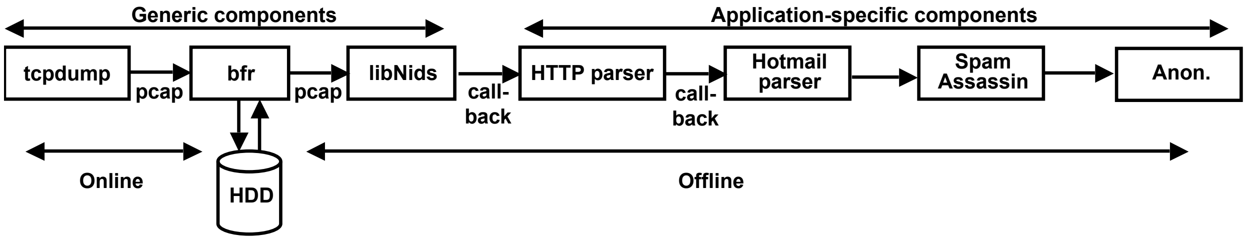

Figure 4. ***Flow of trace data through Bunker's modules.*** *The online part of Bunker consists of tcpdump and the bfr buffering module. The offline part of Bunker consists of bfr, libNids, HTTP parser, Hotmail parser, SpamAssassin, and an anonymizer module. Also, tcpdump, bfr, and libNids are generic components to Bunker, wherease HTTP parser, Hotmail parser, SpamAssassin, and the anonymized module are specific to our current application: collecting traces of phishing e-mail.*

receive buffers. All output from tcpdump is sent directly to *bfr*, a Linux non-blocking pipe buffer that buffers data between Bunker's offline and online components. We use multiple memory mapped files residing on the encrypted disks as the bfr buffer and we allocate 380 GB of disk space to it, sufficient to buffer over 8 hours of HTTP traffic for our network. Figure 5 shows how bfr's buffer size varies over time.

Our Bunker deployment at the University of Toronto is able to trace continuously, even with an unoptimized offline component. This is because of the cyclical nature of network traffic (e.g., previous studies showed that university traffic is 1.5 to 2 times lower on a weekend day than on a week day [42, 50]). This allows the offline component to catch up with the online component during periods of low load, such as nights and weekends. In general, Bunker can only trace continuously if the buffer drains completely at least once during the week. If the peak buffer size during a week day is $p$ and Bunker's offline component leaves $\Delta$ unprocessed at the end of a week day (see Figure 5), Bunker is able to trace continuously if the following two conditions hold:

**1.** Bunker's buffer size is larger than $4 \times \Delta + p$, or the amount of unprocessed data after four consecutive week days plus the peak traffic on the fifth week day;

**2.** During the weekend, Bunker's offline component can catch up to the online component by at least $5 \times \Delta$ of the unprocessed data in the buffer.

The tracing application we built using Bunker gathers traces of phishing e-mails received by Hotmail users at the University of Toronto. The offline trace analysis component performs five tasks: 1) reassembling packets into TCP streams; 2) parsing HTTP; 3) parsing Hotmail; 4) running SpamAssassin over the Hotmail e-mails, and 5) anonymizing output. To implement each of these tasks, we wrote simple Python and Perl scripts that made extensive use of existing libraries and tools.

For TCP/IP reconstruction, we used *libNids* [48], a C library that runs the TCP/IP stack from the Linux 2.0 kernel in user-space. libNids supports reassembly of both IP fragments and TCP streams. Both the HTTP and the Hotmail parsers are written in Python version 2.5. We used a wrapper for libNids in Python to interface with our HTTP parsing code. Whenever a TCP stream is assembled, libNids calls a Python function that passes on the content to the HTTP and Hotmail parsers. The Hotmail parser passes the bodies of the e-mail messages to SpamAssassin (written in Perl) to utilize its spam and phishing detection algorithms. The output of SpamAssassin is parsed and then added to an internal object that represents the Hotmail message. This object is then serialized as a Python "pickled" object before it is transferred to the anonymization engine. We used an HTTP anonymization policy similar to the one described in [35]. We took two additional steps towards ensuring that the anonymization policy is correctly specified and implemented: (1) we performed a code review of the policy and its implementation, and (2) we made the policy and the code available to the University of Toronto's network operators encouraging them to inspect it.

### 3.6 Debugging

Debugging a closed-box environment is challenging because an attacker could use the debugging interface to extract sensitive internal state from the system. Despite this restriction, we found the development of Bunker's analysis software to be relatively easy. Our experience found the off-the-shelf analysis code we used in Bunker to be well tested and debugged. We used two additional techniques for helping to debug Bunker's analysis code. First, we tested our software extensively in the lab against synthetic traffic sources that do not pose any privacy risks. To do this, we booted Bunker into a special diagnostic mode that left I/O devices (such as the keyboard and monitor) enabled. This configuration allowed us to easily debug the system and patch the analysis software without rebooting.

Second, we ensured that every component of our analysis software produced diagnostic logs. These logs were sent from the closed-box VM to the open-box VM using

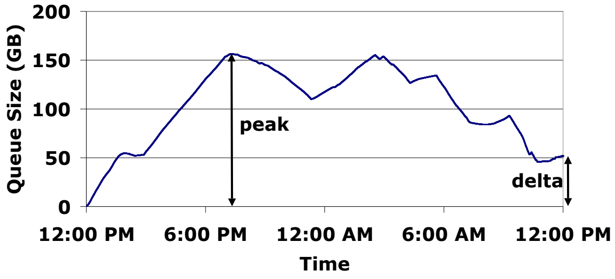

Figure 5. ***The size of bfr's buffer over time.*** *While the queue size increases during the day, it decreases during night when there is less traffic. At the end of this particular day, Bunker's offline component still had 50GB of unprocessed raw trace left in the buffer.*

the same interface as the anonymized trace. They proved helpful in shedding light on the "health" of the processes inside the closed-box VM. We were careful to ensure that no sensitive data could be written to the log files in order to preserve trace data privacy.

## 4 The Benefits of Bunker

This section presents the benefits offered by Bunker's architecture.

### 4.1 Privacy Benefits

Unlike offline anonymization, our approach does not allow network administrators or researchers to work directly with sensitive data at any time. Because unanonymized trace data cannot be directly accessed, it cannot be produced 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 more powerful than those offered by online anonymization. Bunker's anonymization key is stored within the closed-box VM, which prevents anyone from accessing it. While online anonymization tracing systems are typically careful to avoid writing unanonymized data to stable storage, they generally do not protect the anonymization key against theft by an adversary with the ability to login to the machine.

### 4.2 Software Engineering Benefits

When an encrypted disk is used to store the raw network trace for later processing, the trace analysis code is free to run offline at slower than line speeds. Bunker supports two models for tracing. In *continuous tracing*, the disk acts as a large buffer, smoothing the traffic's bursts and its daily cycles. To trace network traffic continuously, Bunker's offline analysis code needs to run fast enough for the *average* traffic rate, but it need not keep up with the *peak* traffic rate. Bunker also supports deferred trace analysis, where the length of the tracing period is limited by the amount of disk storage, but there are no constraints on the performance of the offline trace analysis code. In contrast, online anonymization tracing systems process data as it arrives and therefore must handle peak traffic in real-time.

Bunker's flexible performance requirements let the developer use managed languages and sophisticated libraries when creating trace analysis software. As a result, its code is both easier to write and less likely to contain bugs. The phishing analysis application using Bunker was built by one graduate student in less than two months, including the time spent configuring the closed-box environment (a one-time cost with Bunker). This development effort contrasts sharply with our experience developing tracing systems with online anonymization. To improve performance, these systems required developers to write carefully optimized code in low-level languages using sophisticated data structures. Bunker lets us use Python scripts to parse HTTP, a TCP/IP reassembly library, and Perl scripts running SpamAssassin.

### 4.3 Fault Handling Benefits

One serious drawback of most online trace analysis techniques is their inability to cope gracefully with bugs in the analysis software. Often, these are "corner-case" bugs that arise in abnormal traffic patterns. In many cases researchers and network operators would prefer to ignore these abnormal flows and continue the data gathering process; however, if the tracing software crashes, all data would be lost until the system can be restarted. This could 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 crashes are more likely to affect long-lived than short-lived flows.

Bunker can better cope with bugs because its online and offline components are fully decoupled. This provides a number of benefits. First, Bunker's online trace collection software is simple because it only captures packets and loads them in RAM (encryption is handled automatically at the file system layer). Its simplicity and size make it easy to test extensively. Second, the online software need not change even when the type of trace analysis being performed changes. Third, the offline trace analysis software also becomes much simpler because it need not be heavily optimized to run at line speed. Unoptimized software tends to have a simpler program structure and therefore fewer bugs. Simpler program structure also makes it easier to recover from bugs when they do arise. Finally, a decoupled architecture makes it possible to identify the flow that caused the error in the trace analyzer, filter out that flow from the buffered raw trace, and restart the trace analyzer so that it never sees that flow as input and thereby avoids the bug entirely. Section 7 quantifies the effect of this improved fault handling on the number of flows that are dropped due to a parsing bug.

## 5 Security Attacks

Bunker's design is inspired by Terra, a VM-based platform for trusted computing [20]. Both Terra and Bunker protect sensitive data by encapsulating it in a closed-box VM with deliberately restricted I/O interfaces. The security of such architectures does not rest on the size of the trusted computing base (TCB) but on whether an attacker can exploit a vulnerability through the system's narrow interfaces. Even if there is a vulnerability in the OS running in the closed-box VM, Bunker remains secure as long as attackers cannot exploit the vulnerability through the restricted channels. In our experience, ISPs have found Bunker's security properties a significant step forward in protecting users privacy when tracing.

Attacks on Bunker can be categorized into three classes. The first are those that attempt to subvert the narrow interfaces of the closed-box VM. A successful attack on these interfaces exposes the closed-box VM's internals. The second class are physical attacks, in which the attacker tampers with Bunker's hardware. The third possibility are attacks whereby Bunker deliberately allows network traffic into the closed-box VM: an attacker could try to exploit a vulnerability in the trace analysis software by injecting traffic in the network being monitored. We now examine each attack type in greater detail.

### 5.1 Attacking the Restricted Interfaces of the Closed-Box VM

There are three ways to attack the restricted interfaces of the closed-box VM: 1) subverting the isolation provided by the VMM to access the memory contents of the closed-box VM; 2) exploiting a security vulnerability in one of the system's drivers; and 3) attacking the closed-box VM directly using the one-way-initiation interface between the closed and open-box VMs.

#### 5.1.1 Attacking the VMM

We use a VMM to enforce isolation between software components that need access to sensitive data and those that do not. Bunker's security rests on the assumption that VMM-based isolation is hard to attack, an assumption made by many in industry [23, 47] and the research community [20, 11, 6, 43]. There are other approaches we could have used to confine sensitive data strictly to the pre-loaded analysis software. For example, we could have used separate physical machines to host the closed and open box systems. Alternatively, we could have relied on a kernel and its associated isolation mechanisms, such as processes and file access controls. However, VM-based isolation is generally thought to provide stronger security than process-based isolation because VMMs are small enough to be rigorously verified and export only a very narrow interface to their VMs [6, 7, 29]. In contrast, kernels are complex pieces of software that expose a rich interface to their processes.

#### 5.1.2 Attacking the Drivers

Drivers are among the buggiest components of an OS [8]. Security vulnerabilities in drivers let attackers bypass all access restrictions imposed by the OS. Systems without an IOMMU are especially susceptible to buggy drivers because they cannot prevent DMA-capable hardware from accessing arbitrary memory addresses. Many filesystem drivers can be exploited by carefully crafted filesystems [53]. Thus, if Bunker were to automount inserted media, an attacker could compromise the system by inserting a CDROM or USB memory device with a carefully crafted filesystem image.

Bunker addresses such threats by disabling all drivers (including the monitor, mouse, and keyboard) except these four: 1) the network capture card driver, 2) the hard disk driver, 3) the driver for the standard NIC used to enable networking in the open-box VM, and 4) the driver for the virtual interfaces used between the closed-box and open-box VMs. In particular, we were careful to disable external storage device support (i.e. CDROM, USB mass storage) and USB support.

We examined each of these drivers and believe that none can be exploited to gain access to the closed-box. First, the network capture card loads incoming network traffic via one of the drivers left enabled in Domain0. This capture card, a special network monitoring card made by Endace (DAG 4.3GE) [17], cannot be used for two-way communication. Thus, an attacker cannot gain remote access to the closed-box solely through this network interface. The second open communication channel is the SCSI controller driver for our hard disks. This is a generic Linux driver, and we checked the Linux kernel mailing lists to ensure that it had no known bugs. The third open communication channel, the NIC used by the open-box VM, remains in the closed-box VM because Xen's design places all hardware drivers in Domain0. We considered mapping this driver directly into DomainU, but doing so would create challenging security issues related to DMA transfers that are best addressed with specialized hardware support (SecVisor [43] discusses these issues in detail). Instead, we use firewall rules to ensure that all outbound communication on this NIC originates from the open-box VM. As with the SCSI driver, this is a generic Linux gigabit NIC driver, and we verified that it had no known bugs. The final open communication channel is constructed by installing a virtual NIC in both the closed-box and open-box VMs and then building a virtual network between them. Typical for most Xen environments, this configuration permits communication across different Domains. As with the SCSI driver, we checked that it had no known security vulnerabilities.

#### 5.1.3 Attacking the One-Way-Initiation Interface

Upon startup, Bunker firewalls the interface between the open-box VM and the closed-box VM using iptables. The rules used to configure iptables dictate that no connections are allowed unless they originate from the closed-box VM (see Figure 3). We re-used a set of rules from an iptables configuration for firewalling home environments found on the Internet.

### 5.2 Attacking Hardware

Bunker protects the closed-box VM from hardware attacks by making it safe-on-reboot. If an attacker turns off the machine to tamper with the hardware (e.g. by removing existing hardware or installing new hardware), the sensitive data contained in the closed-box VM is effectively destroyed. This is because the encryption keys and any unencrypted data are only stored in volatile memory (RAM). Therefore, hardware attacks must be mounted while the system is running. Section 5.1.2 discusses how we eliminated all unnecessary drivers from Bunker; this protects Bunker against attacks relying on adding new system devices, such as USB devices.

Another class of hardware attacks is one in which the attacker attempts to extract sensitive data (e.g., the encryption keys) from RAM. Such attacks can be mounted in many ways. A recent project demonstrated that the contents of today's RAM modules may remain readable even minutes after the system has been powered off [21]. Bunker is vulnerable to such attacks: an attacker could try to extract the encryption keys from memory by removing the RAM modules from the tracing machine and placing them into one configured to run key-searching software over memory on bootup [21]. Another approach is to attach a bus monitor to observe traffic on the memory bus. Preventing RAM-based attacks requires specialized hardware, which we discuss below. Yet another way is to attach a specialized device, such as certain Firewire devices, that can initiate DMA transfers without any support from software running on the host [37, 14]. Preventing this attack requires either 1) disabling the Firewire controller or 2) support from an IOMMU to limit which memory regions can be accessed by Firewire devices.

**Secure Co-processors Can Prevent Hardware Attacks:** A secure co-processor contains a CPU packaged with a moderate amount of non-volatile memory enclosed in a tamper-resistant casing [44]. A secure co-processor would let Bunker store the encryption and anonymization keys, the unencrypted trace data and the code in a secure environment. It also allows the code to be executed within the secure environment.

**Trusted Platform Modules (TPMs) Cannot Prevent Hardware Attacks:** Unfortunately, the use of TPMs would not significantly help Bunker survive hardware attacks. The limited storage and execution capabilities of a TPM cannot fully protect encryption keys and other sensitive data from an adversary with physical access [21]. This is because symmetric encryption and decryption are not performed directly by the TPM; these operations are still handled by the system's CPU. Therefore, the encryption keys must be exposed to the OS and stored in RAM, making them subject to the attack types mentioned above.

### 5.3 Attacking the Trace Analysis Software

An attacker could inject carefully crafted network traffic to exploit a vulnerability in the trace analysis software, such as a buffer overflow. Because this software does not run as root, such attacks cannot disable the narrow interfaces of the closed-box; the attacker needs root privileges to alter the OS drivers or the iptable's firewall rules. Nevertheless, such an attack could obtain access to sensitive data, skip the anonymization step, and send captured data directly to the open-box VM through the one-way-initiation interface.

While possible, such attacks are challenging to mount in practice for two reasons. First, Bunker's trace analysis software combines C (e.g., tcpdump plus a TCP/IP reconstruction library, which is a Linux 2.0 networking stack running in user-space), Python, and Perl. The C code is well-known and well-tested, making it less likely to have bugs that can be remotely exploited by injecting network traffic. Bunker's application-level parsing code is written in Python and Perl, two languages that are resistant to buffer overflows. In contrast, online anonymizers write all their parsing code in unmanaged languages (e.g., C or C++) in which it is much harder to handle code errors and bugs.

Second, a successful attack would send sensitive data to the open-box VM. The attacker must then find a way to extract the data from the open-box VM. To mitigate this possibility, we firewall the open-box's NIC to reject any traffic unless it originates from our own private network. Thus, to be successful, an attacker must not only find an exploitable bug in the trace analysis code but must also compromise the open-box VM through an attack that originates from our private network.

## 6 Operational Issues

At boot time, Bunker's bootloader asks the user to choose between two configurations: an ordinary one and a restricted one. The ordinary configuration loads a typical Xen environment with all drivers enabled. We use this environment only to prepare a tracing experiment and to configure Bunker; we never gather traces in it because it offers no privacy benefits. To initiate a tracing experiment, we boot into the restricted environment. When booting into this environment, Bunker's display and keyboard freeze because no drivers are being loaded. In this configuration, we use the open NIC to log in to the open-box VM where we can monitor the anonymized traces received through the one-way-initiation interface. These traces also contain meta-data about the health of the closed-box VM, including a variety of counters (such as packets received, packets lost, usage of memory, and amount of free space on the encrypted disk).

Network studies often need traces that span weeks, months, or even years. The closed-box nature of Bunker and its long-term use raise the possibility of the following operational attack: an intruder gains physical access to Bunker, reboots it, and sets it up with a fake restricted environment that behaves like Bunker's restricted environment but uses encryption and anonymization keys known to the intruder. This attack could remain undetected by network operators. From the outside, Bunker seems to have gathered network traces continuously.

To prevent this attack, Bunker could generate a public/private key-pair upon starting the closed-box VM. The public key would be shared with the network operator who saves an offline copy, while the private key would never be released from the closed-box VM. To verify that Bunker's code has not been replaced, the closed-box VM would periodically send a heartbeat message through the one-way-initiation interface to the open-box. The heartbeat message would contain the experiment's start time, the current time, and additional counters, all signed with the private key to let network operators verify that Bunker's original closed-box remains the one currently running. This prevention mechanism is not currently implemented.

## 7 Evaluation

This section presents a three-pronged evaluation of Bunker. First, we measure the performance overhead introduced by virtualization and encryption. Second, we evaluate Bunker's software engineering benefits when compared to online tracing tools. Third, we conduct an experiment to show Bunker's fault handling benefits.

### 7.1 Performance Overhead

To evaluate the performance overhead of virtualization and encryption, we ran *tcpdump* (i.e., Bunker's online component) to capture all traffic traversing a gigabit link and store it to disk. We measured the highest rate of traffic tcpdump can capture with no packet losses under four configurations: standalone, running in a Xen VM, running on top of an encrypted disk with *dm-crypt* [41], and running on top of an encrypted disk in a Xen VM.

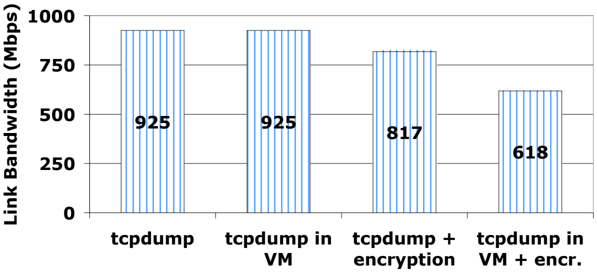

Figure 6. ***Performance overhead of virtualization and encryption:*** *We measured the rate of traffic that tcpdump can capture on our machine with no packet losses under four configurations: standalone, running in a Xen VM, running on top of an encrypted file system, and running on top of an encrypted file system in a Xen VM. All output captured by tcpdump was written to the disk.*

Our tracing host is a dual Intel Xeon 3.0GHz with 4 GB of RAM, six 150 GB SCSI hard-disk drives, and a DAG 4.3GE capture card. We ran Linux Debian 4.0 (etch), kernel version 2.6.18-4 and attached the tracer to a dedicated Dell PowerConnect 2724 gigabit switch with two other commodity PCs attached. One PC sent constant bit-rate (CBR) traffic at a configurable rate to the other; the switch was configured to mirror all traffic to our tracing host. We verified that no packets were being dropped by the switch.

Figure 6 shows the results of this experiment. The first bar shows that we capture 925 Mbps when running tcpdump on the bare machine with no isolation. The limiting factor in this case is the rate at which our commodity PCs can exchange CBR traffic; even after fine tuning, they can exchange no more than 925 Mbps on our gigabit link. The second bar shows that running tcpdump inside the closed-box VM has no measurable effect on the capture rate because the limiting factor remains our traffic injection rate. When we use the Linux dm-crypt module for encryption, however, the capture rate drops to 817 Mbps even when running on the bare hardware: the CPU becomes the bottleneck when running the encryption module. Combining both virtualization and encryption shows a further drop in the capture rate, to 618 Mbps. Once the CPU is fully utilized by the encryption module, the additional virtualization costs become apparent.

Our implementation of Bunker can trace network traffic of up to 618 Mbps with no packet loss. This is sufficiently fast for the tracing scenario that our university requires. While the costs of encryption and virtualization are not negligible, we believe that these overheads will decrease over time as Linux and Xen incorporate further optimizations to their block-level encryption and virtualization software. At the same time, CPU manufacturers have started to incorporate hardware acceleration for AES encryption (i.e., similar to what dm-crypt uses) [46].

## 7.2 Software Engineering Benefits

As previously discussed, Bunker offers significant software engineering benefits over online network tracing systems. Figure 7 shows the number of lines of code for three network tracing systems that perform HTTP parsing, all developed by this paper's authors. The first two systems trace HTTP traffic at line speeds. The first system was developed from scratch by two graduate students over the course of one year. The second system was developed by one graduate student in nine months; this system was built on top of CoMo, a packet-level tracing system developed by Intel Research [22]. Bunker is the third system; it was developed by one student in two months. As Figure 7 shows, Bunker's codebase is an order of magnitude smaller than the others. Moreover, we wrote only about one fifth of Bunker's code; the remainder was re-used from libraries.

Bunker's smaller and simpler codebase comes at a cost in terms of its offline component's performance. Figure 8 shows the time elapsed for Bunker's online and offline components to process a 5 minute trace of HTTP traffic. The trace contains 4.5 million requests, or about 15,000 requests per second, that we generated using *httpperf*. In practice, very few traces contain that many HTTP requests per second. While the online component runs only tcpdump storing data to the disk, the offline component performs TCP/IP reconstruction, parses HTTP, and records the HTTP headers before copying the trace to the open-box VM. The offline component spends 20 minutes and 28 seconds processing this trace. Clearly, Bunker's ease of development comes at the cost of performance, as we did not optimize the HTTP parser at all. The privacy guarantees of our isolated environment grant us the luxury of re-using existing software components even though they do not meet the performance demands of online tracing.

## 7.3 Fault Handling Evaluation

In addition to supporting fast development of different tracing experiments, Bunker handles bugs in the tracing software robustly. Upon encountering a bug, Bunker marks the offending flow as "erroneous" and continues processing traffic without having to restart. To illustrate the benefits of this fault handling approach, we performed the following experiment. We used Bunker on a Saturday to gather a 20 hour trace of the HTTP traffic our university exchanges with the Internet. This trace contained over 5.2 million HTTP flows. We artificially injected a parsing bug in one packet out of 100,000 (corresponding to a parsing error rate of 0.001%). Upon encountering this bug, Bunker stops parsing the erroneous HTTP flow and continues with the remaining flows. We compare Bunker to an online tracer that would crash upon encountering a bug and immediately restart. This would result in the online tracer dropping all concurrent flows (we refer to this as "collateral damage"). This experiment assumes an idealized version of an online tracer that restarts instantly; in practice, it takes tens of seconds to restart an online tracer's environment losing even more ongoing flows. Figure 9 illustrates the difference in the fraction of flows affected. While our bug is encountered in only 0.08% of the flows, it affects an additional 31.72% of the flows for an online tracing system. Not one of these additional flows is affected by the bug when Bunker performs the tracing.

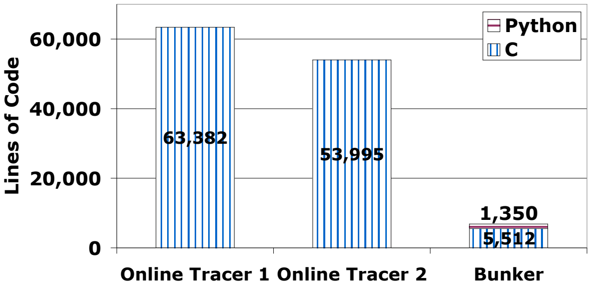

Figure 7. ***Lines of Code in three systems for gathering HTTP traces:*** *The first system was developed from scratch by two graduate students in one year. The second system, an extension of CoMo [22], was developed by one graduate student in nine months; we included CoMo's codebase when counting the size of this system's codebase. The third system, Bunker, was developed by one student in two months.*

# 8 Legal Background

This section presents legal background concerning the issuing of subpoenas for network traces in the U.S. and Canada and discusses legal issues inherent in designing and deploying data-hiding tracing platforms[2].

## 8.1 Issuing Subpoenas for Data Traces

U.S. law has two sets of requirements for obtaining a data trace that depend on when the data was gathered. For data traces gathered in the past 180 days, the government needs a *mere subpoena*. Such subpoenas are obtained from a federal or state court with jurisdiction over the offense under investigation. Based on our conversations with legal experts, obtaining a subpoena is relatively simple in the context of a lawsuit. A defendant

[2] Any mistakes in our characterization of the U.S. or Canadian legal systems are the sole responsibility of the authors and not the lawyers we consulted during this research project.

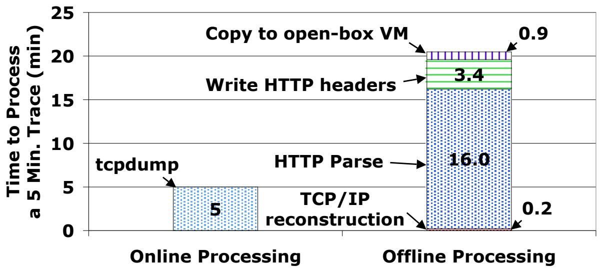

Figure 8. ***Online vs. Offline processing speed:*** *The time spent processing a five minute HTTP trace by Bunker's online and offline components, respectively.*

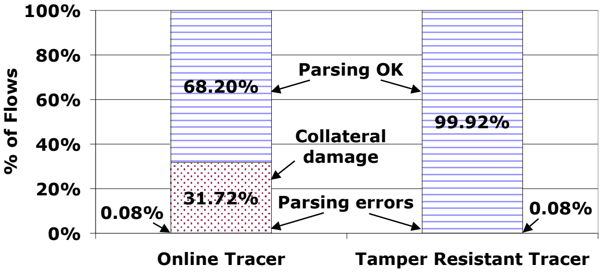

Figure 9. ***Fraction of flows affected by a bug in an online tracer versus in Bunker:*** *A bug crashing an online tracer affects all flows running concurrent with the crash. Instead, Bunker handles bugs using exceptions affecting only the flows that triggered the bug.*

(e.g., the ISP) could try to quash the subpoena if compliance would be unreasonable or oppressive.

For data gathered more than 180 days earlier, a government entity needs a warrant under Title 18 United States Code 2703(d) from a federal or state court with appropriate jurisdiction. The government needs to present "specific and articulable facts showing that there are reasonable grounds to believe that the contents of a wire or electronic communication, or the records or other information sought, are relevant and material to an ongoing criminal investigation." The defendant can quash the subpoena if the information requested is "unusually voluminous in nature" or compliance would cause undue burden. Based on our discussions with legal experts, the court would issue such a warrant if it determines that the data is relevant and not duplicative of information already held by the government entity.

In Canada, a subpoena is sufficient to obtain a data trace regardless of the data's age. In 2000, the Canadian government passed the Personal Information Protection and Electronic Documents Act (PIPEDA) [33], which enhances the users' rights to privacy for their data held by private companies such as ISPs. However, Section 7(3)(c.1) of PIPEDA indicates that ISPs must disclose personal information (including data traces) if they are served with a subpoena or even an "order made by a court, person or body with jurisdiction to compel production of information". In a recent case, a major Canadian ISP released personal information to the local police based on a letter that stated that "the request was done under the authority of PIPEDA" [32]. A judge subsequently found that prior authorization for this information should have been obtained, and the ISP should not have disclosed this information. This case illustrates the complexity of the legal issues ISPs face when they store personal information (e.g., raw network traces).

### 8.2 Developing Data-Hiding Technology

In our discussions with legal experts, we investigated whether it is legal to develop and deploy a data-hiding network tracing infrastructure (such as Bunker). While there is no clear answer to this question without legal precedent, we learned that the way to evaluate this question is to consider the purpose and potential uses for the technology in question. In general, it is legal to deploy a technology that has many legitimate uses but could also enable certain illegitimate uses. Clearly, technologies whose primary use is to enable or encourage users to evade the law are not legal. A useful example to illustrate this distinction is encryption technology. While encryption can certainly be used to enable illegal activities, its many legitimate uses make development and deployment of encryption technologies legal. In the context of network tracing, protecting users' privacy against accidental loss or mismanagement of the trace data is a legitimate purpose.

## 9 Related Work

Bunker draws on previous work in network tracing systems, data anonymizing techniques, and virtual machine usage for securing systems. We summarize this previous work and then we describe two systems built to protect access to sensitive data, such as network traces.

### 9.1 Network Tracing Systems

One of the earliest network tracing systems was Httpdump [51], a tcpdump extension that constructs a log of HTTP requests and responses. Windmill [30] developed a custom packet filter that facilitates the building of specific network analysis applications; it delivers captured packets to multiple filters using dynamic code generation. BLT [18], a network tracing system developed specifically to study HTTP traffic, supports continuous online network monitoring. BLT does not use online anonymization; instead, it records raw packets directly to disk. More recently, CoMo [22] was designed to allow independent parties to run multiple ongoing trace analysis modules by isolating them from each other. With CoMo, anonymization, whether online or offline, must be implemented by each module's owner. Unlike these systems, Bunker's design was motivated by the need to protect the privacy of network users.

### 9.2 Anonymization Techniques

Xu et al. [52] implemented a prefix-preserving anonymization scheme for IP addresses, i.e., addresses with the same IP prefix share the same prefix after anonymization. Pang et al. [35] designed a high-level language for specifying anonymization policies, allowing researchers to write short policy scripts to express trace transformations. Recent work has shown that traces can still leak private information even after they are anonymized [34], prompting the research community to propose a set of guidelines and etiquette for sharing data traces [1]. Bunker's goal is to create a tracing system that makes it easy to develop trace analysis software while ensuring that no raw data can be exposed from the closed-box VM. Bunker does not protect against faulty anonymization policies, nor does it ensure that anonymized data cannot be subject to the types of attacks described in [34].

### 9.3 Using VMs for Making Systems Secure

An active research area is designing virtual machine architectures that are secure in the face of attacks. Several solutions have been proposed, including: using tamper-resistant hardware [28, 20]; designing VMMs that are small enough for formal verification [25, 40]; using programming language techniques to provide memory safety and control-flow integrity in commodity OS'es [26, 12]; and using hardware memory protection to provide code integrity [43]. While these systems attempt to secure a general purpose commodity OS, Bunker was designed only to secure tracing software. As a result, its interfaces are simple and narrow.

### 9.4 Protecting Access to Sensitive Data

Packet Vault [3] is a network tracing system that captures packets, encrypts them, and writes them to a CD. A newer system design tailored for writing the encrypted traces to tape appears in [2]. Packet Vault creates a permanent record of all network traffic traversing a link. Its threat model differs from Bunker's in that there is no attempt to secure the system against physical attacks.

Armored Data Vault [24] is a system that implements access control to previously collected network traces, by using a secure co-processor to enforce security in the face of malicious attackers. Like Bunker, network traces are encrypted before being stored. The encryption key and any raw data are stored inside the secure co-processor. Bunker's design differs from Armored Data Vault's in three important ways. First, Bunker's goal is limited to trace anonymization only and not to implementing access control policies; this lets us use simple, off-the-shelf anonymization code to minimize the likelihood of bugs present in the system. Second, Bunker destroys the raw data as soon as it is anonymized; the Armored Data Vault stores its raw traces permanently while enforcing the data access policy. Finally, Bunker uses commodity hardware that can run unmodified off-the-shelf software. Instead, the authors of the Armored Data Vault had to port their code to accommodate the specifics of the secure co-processor, a process that required effort and affected the system's performance [24].

## 10 Conclusions

This paper presents Bunker, a network tracing architecture that combines the performance and software engineering benefits of offline anonymization with the privacy offered by online anonymization. Bunker uses a closed-box and safe-on-reboot architecture to protect raw trace data against a large class of security attacks, including physical attacks to the system. In addition to its security benefits, our architecture improves ease of development: using Bunker, one graduate student implemented a network tracing system for gathering anonymized traces of Hotmail e-mail in less than two months.

Our evaluation shows that Bunker has adequate performance. We show that Bunker's codebase is an order of magnitude smaller than previous network tracing systems that perform online anonymization. Because most of its data processing is performed offline, Bunker also handles faults more gracefully than previous systems.

**Acknowledgments**: We are grateful to Gianluca Iannaccone and Intel Cambridge for donating the hardware for Bunker. We thank John Dunagan, Steve Gribble, Steve Hand (our shepherd), Srikanth Kandula, Andy Warfield, and the anonymous reviewers for their insightful feedback.

## References

[1] M. Allman and V. Paxson. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the Internet Measurement Conference (IMC)*, San Diego, CA, October 2007.

[2] C. J. Antonelli, K. Coffman, J. B. Fields, and P. Honeyman. Cryptographic wiretapping at 100 megabits. In *16th SPIE Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, FL, April 2002.

[3] C. J. Antonelli, M. Undy, and P. Honeyman. The packet vault: Secure storage of network data. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA, April 1999.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Bolton Landing, NY, 2003.

[5] BBC News. Brown apologises for records loss, November 2007. `http://news.bbc.co.uk/1/hi/uk_politics/7104945.stm`.

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

[7] P. M. Chen and B. D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, Elmau, Germany, May 2001.

[8] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating system errors. In *Proc. of the 18th ACM Symposium on Operating System Principles (SOSP)*, pages 73–88, Banff, Canada, October 2001.

[9] M. P. Collins and M. K. Reiter. Finding p2p file-sharing using coarse network behaviors. In *Proceedings of ESORICS*, 2006.

[10] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized traces. In *Proceedings of the USENIX Security Symposium*, Boston, MA, August 2007.

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

[12] J. Criswell, A. Lenharth, D. Dhurjati, and V. Adve. Secure virtual architecture: A safe execution environment for commodity operating systems. In *21st ACM SOSP*, Stevenson, WA, Oct 2007.

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

[14] M. Dornseif. 0wned by an ipod, 2004. http://md.hudora.de/presentations/firewire/PacSec2004.pdf.

[15] N. Duffield, C. Land, and M. Thorup. Charging from sampled network usage. In *Proceedings of Internet Measurement Workshop (IMW)*, San Francisco, CA, November 2001.

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

[17] Endace. Dag network monitoring cards – ethernet, 2008. http://www.endace.com/our-products/dag-network-monitoring-cards/etherne%t.

[18] A. Feldmann. BLT: Bi-layer tracing of HTTP and TCP/IP. In *9th International World Wide Web Conference*, Amsterdam, Holland, May 2000.

[19] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: methodology and experience. *IEEE/ACM Transactions on Networking (TON)*, 9(3):265–280, 2001.

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

[21] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, July 2008.

[22] G. Iannaccone. CoMo: An open infrastructure for network monitoring – research agenda. Technical report, Intel Research, 2005.

[23] IBM. sHype – Secure Hypervisor. http://www.research.ibm.com/secure_systems_department/projects/hypervis%or/.

[24] A. Iliev and S. Smith. Prototyping an armored data vault: Rights management on big brother's computer. In *Privacy-Enhancing Technologies*, San Francisco, CA, April 2002.

[25] K. Kaneda. Tiny virtual machine monitor, 2006. http://web.yl.is.s.u-tokyo.ac.jp/~kaneda/tvmm/.

[26] V. Kiriansky, D. Bruening, and S. P. Amarasinghe. Secure execution via program shepherding. In *11th USENIX Security Symposium*, San Francisco, CA, August 2002.

[27] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of 2004 Conference on Applications, Technologies, Architectures, and Protocols for computer communications (SIGCOMM)*, Portland, OR, September 2004.

[28] D. Lie, C. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware. In *19th ACM SOSP*, Bolton Landing, NY, October 2003.

[29] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proc. of the Workshop on Virtual Computer Systems*, Cambridge, MA, 1973.

[30] G. R. Malan and F. Jahanian. An extensible probe architecture for network protocol performance measurement. In *Proceedings of the 1998 Conference on Applications, Technologies, Architectures, and Protocols for computer communications (SIGCOMM)*, Vancouver, BC, September 1998.

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

[32] Mondaq. Canada: Disclosure of personal information without consent pursuant to lawful authority, 2007. http://www.mondaq.com/article.asp?article_id=53904.

[33] Office of the Privacy Commissioner of Canada. Personal information protection and electronic documents act, 2000. http://www.privcom.gc.ca/legislation/02_06_01_01_e.asp.

[34] 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.

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

[36] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proc. of IMC)*, Taormina, Italy, October 2004.

[37] P. Panholzer. Physical security attacks on windows vista. Technical report, SEC Consult Unternehmensberatung GmbH, 2008. http://www.sec-consult.com/fileadmin/Whitepapers/Vista_Physical_Attacks%.pdf.

[38] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot. Long-term forecasting of internet backbone traffic: Observations and initial models. In *Proc. of INFOCOM*, San Francisco, CA, April 2003.

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

[40] R. Russell. Lguest: The simple x86 hypervisor, 2008. http://lguest.ozlabs.org/.

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

[42] 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.

[43] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *21st ACM SOSP*, October 2007.

[44] J. D. Tygar and B. Yee. Dyad: A system for using physically secure coprocessors. In *IP Workshop Proceedings*, 1994.

[45] R. Vasudevan and Z. M. Mao. Reval: A tool for real-time evaluation of ddos mitigation strategies. In *Proceedings of the 2006 Usenix Annual Technical Conference*, 2006.

[46] VIA. Via padlock security engine. http://www.via.com.tw/en/initiatives/padlock/hardware.jsp.

[47] VMware. VMware VMsafe Security Technology. http://www.vmware.com/overview/security/vmsafe.html.

[48] R. Wojtczuk. libNids, 2008. http://libnids.sourceforge.net/.

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

[50] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based analysis of web-object sharing and caching. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS)*, Boulder, CO, October 1999.

[51] R. Wooster, S. Williams, and P. Brooks. HTTPDUMP network HTTP packet snooper, April 1996. http://ei.cs.vt.edu/~succeed/96httpdump/.

[52] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix-preserving IP traffic trace anonymization. In *1st ACM SIGCOMM IMW Workshop*, 2001.

[53] J. Yang, C. Sar, P. Twohey, C. Cadar, and D. Engler. Automatically generating malicious disks using symbolic execution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 243–257, Oakland, CA, May 2006.

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