Mikro-SINA—Hands-on Experiences with the Nizza Security Architecture

Christian Helmuth · Alexander Warg · Norman Feske

Technische Universität Dresden
Department of Computer Science
Operating Systems Group
{helmuth,warg,feske}@os.inf.tu-dresden.de

Abstract

Key components of today's network communication infrastructure of German embassies, called Secure Inter-Network Architecture (SINA), are based on the GNU/Linux platform. However, the Linux kernel with over 500,000 lines of code is rather complex and must be fully trusted to maintain confidentiality and integrity of the processed data. In this paper, we present an approach to reduce the complexity of the trusted computing base of a VPN gateway by an order of magnitude while maintaining full functionality. We show how to safely reuse untrusted legacy software on trusted platforms and thus, rapidly increase overall security with low engineering costs.

1 Introduction

A Virtual Private Network (VPN) connects islands of trusted private networks over untrusted links and especially over the Internet. This enables distant network nodes (e. g., portable devices) to securely communicate with a company's server via a virtual network. In IPSec-based VPNs, all traffic transported over the Internet is secured by crypthographic means as specified in the IPSec standard. This ensures the protection of sensitive information against unauthorized inspection and manipulation. The VPN gateway implements the needed security mechanisms and acts as the guard at the border between networks of different trust/security levels (e. g., between the private network and the untrusted public Internet).

The majority of current VPN implementations are based on monolithic operating systems. For example, the Secure Inter-Network Architecture (SINA) uses Linux for gateways and end-user systems. In monolithic operating systems, the IPSec implementation is integrated in the kernel and closely interwoven with other components of the kernel such as the network subsystem. Thus, bugs in the kernel code or successful penetration of the complex monolithic kernel can compromise the security-relevant functions.

The following example emphasizes the severity of the situation: In Linux 2.4, 70 percent of the kernel code are device drivers with an error probability 7 times higher than

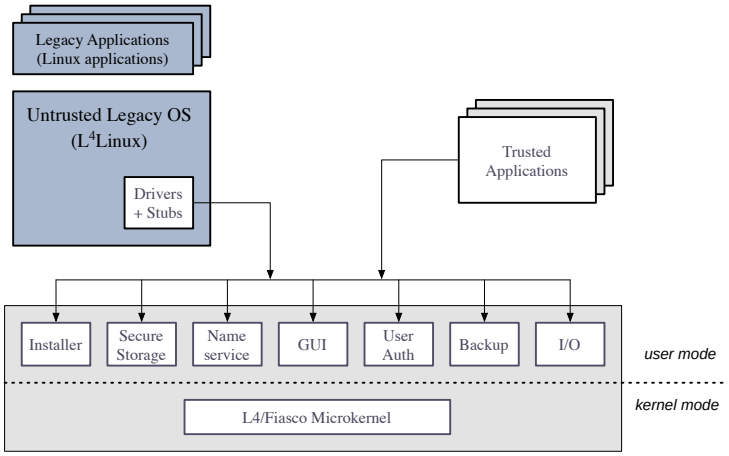

Fig. 1: Nizza Security Architecture. The picture illustrates the overall architecture of Nizza. The box at the bottom contains the microkernel and the basic trusted services.

in other kernel modules [5]. This huge amount of code has unrestricted access to all data structures and functions of the kernel.

We aim at reducing the complexity of the trusted software of the SINA VPN gateway by an order of magnitude. This will enable cost-efficient high-level evaluation and application in high-assurance scenarios. Therefore, we use components and concepts of the Nizza security architecture that shall be introduced in the following section. Section 3 explains our VPN-gateway design and implementation in detail, followed by an overview of related work and the conclusion in the final section.

2 Nizza Security Architecture

2.1 Overview

The foundation of Mikro-SINA are concepts and techniques of the Nizza security architecture. In the following, we shortly present the key concepts that are important to understand Mikro-SINA and refer you to the original paper [7] for details. The main goal of Nizza (and also Mikro-SINA) is to keep the trusted computing base (TCB) of security-sensitive applications as small as possible.

The core concept inherent to Nizza is the (fine-grained) isolation of protection domains. To achieve this, the architectural basis of Nizza is the L4/Fiasco microkernel. It provides a small set of features: Address spaces for isolation, message passing primitives, and threads. Device drivers are not part of the kernel and run in private protection domains atop Fiasco. The limitation of the functionality of the microkernel keeps the complexity of the software running in privileged processor mode as small as 15,000 lines of code (LOC) C++. This and the conceptual nonextensibility of the kernel code lead to systems considered more robust and secure.

For the implementation of real-world applications, microkernel primitives alone are insufficient. Therefore, the Nizza architecture includes a layer of trusted services as depicted in Figure 1. Nizza's trusted service components are small (in complexity means) and are executed within separate protection domains.

An exemplary trusted service of Nizza is a windowing server providing the trusted path from the application to the user. For the deployment of a trusted platform as defined by the Trusted Computing Group (TCG) [1], the architecture also includes an installer that allows to install, control, and attest new trusted applications, a secure storage module, and a user authentication component.

2.2 Trusted Wrappers

An important aspect of Nizza is the reuse of untrusted legacy software to meet the requirement for small additional engineering costs. Legacy software in our context are applications as well as operating systems and device drivers. Our prime example is L^4Linux, a port of the Linux kernel to the L4 microkernel interface [6] that runs as a user-mode program. The design template of trusted wrappers as described by Hohmuth and others in [8] enables us to even use untrusted code for trusted services. Trusted wrappers come in two flavours, the *sandbox wrappers* and the *perimeter wrappers*.

The sandbox wrappers completely encapsulate system components based on untrusted code to prevent leakage of information. Sandboxing makes it possible to have untrusted code work on unprotected data, for example for converting data formats or for running legacy applications on classified data. Components wrapped in this way are comparable to dedicated isolated hosts for security-critical tasks.

On the other hand, perimeter wrapping is used for untrusted components that communicate with the outside world. These components must never see unprotected (unencrypted) data. Such components typically reside at the border of the system (e. g., network connections or storage media). Perimeter wrapping originates from network-security techniques with the prominent example of VPN tunneling.

2.3 Communication Control

In Nizza, all components use cross-domain communication primitives provided by the microkernel. These primitives are message-based inter-process communication (IPC) and shared memory. Therefore, the microkernel is able to restrict interaction of domains and thereby enforces system policies [10]. These policies are defined by trusted services using microkernel primitives.

Address spaces and IPC control effectively isolate threads inside the system. Nevertheless, device drivers may circumvent domain boundaries via Busmaster Direct Memory Access (DMA) devices in current computer systems as described in [7]. The paper proposes several approaches to fully or partially prevent this kind of attacks or faults. Until proper protection from Busmaster DMA attacks is applied, device drivers must be regarded as trusted and reuse of legacy code is impractical. Otherwise, malicious code may compromise sensitive data or system integrity.

Another aspect are covert channels among Nizza components. We did not extensively investigate this issue. However, solutions described in several publications, for example [9], can be adopted to Nizza.

2.4 Terminology for security categories

There is some divergent terminology in the security community about the definitions of the three security categories. In the remainder of the paper, we use the following definitions from [8]:

- **Confidentiality**
  Only authorized users (entity, principal, etc.) can access information (data, programs, etc.).
- **Integrity**
  Either information is current, correct, and complete, or it is possible to detect that these properties do not hold.
- **Availability**
  Data is available when and where an authorized user needs it.

3 Mikro-SINA VPN

In the following we shall describe our design of the VPN gateway and how we reduce the trusted computing base for this application.

3.1 Architecture

The *VPN border* between the private and the untrusted network logically splits the VPN implementation into the following functional parts:

- Processing sensitive (unencrypted) data
- Protecting sensitive data and maintaining policies
- Processing protected (encrypted) data only

We observed that the security-relevant functions of a VPN implementation—data protection and policy enforcement—are only a small fraction of the monolithic kernel (less than 5 %).

The Nizza architecture enables us to extract those IPSec-specific functions and execute them in a separate protection domain. This technique dramatically reduces the vulnerability of this sensitive functionality. We call this IPSec component *Viaduct* as it represents the actual connection point between the private network and the untrusted Internet. Beside the Viaduct, a VPN gateway requires network device drivers, IP packet processing including defragmentation, routing, and other basic networking functions for operation.

All the software components of the VPN gateway excluding the Viaduct must be assigned to either the private or the Internet side. Because both sides require general network functionality, we use two instances of L^{4}Linux running on one machine for providing these functions. Each of these L^{4}Linux instances is isolated through the microkernel and allowed to access one of the two physical network interface connectors (NIC) exclusively[1]. Therefore both instances are not able to communicate directly. The

[1] While the Linux instances are untrusted, the current implementation of Mikro-SINA uses a set of trusted device drivers (running in their own address spaces). We have to trust these drivers because they use unrestricted DMA. See also Section 2.3 and [7].

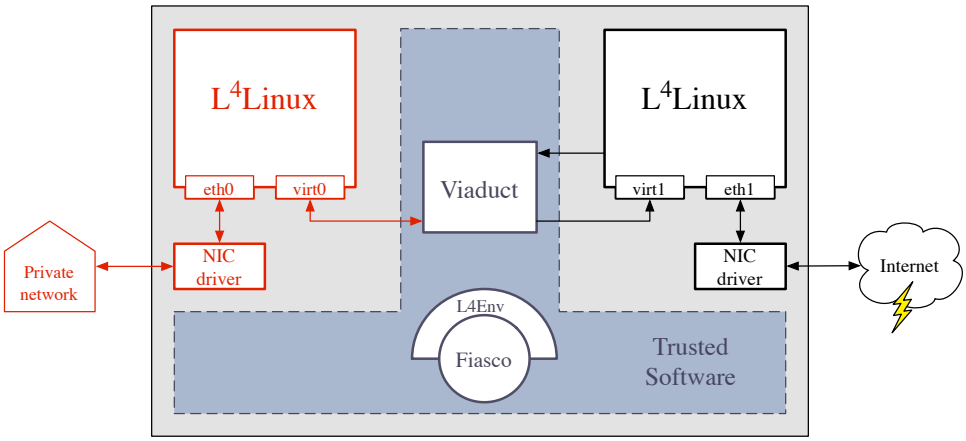

Fig. 2: Mikro-SINA Architecture overview. The illustration shows two L^{4}Linux instances running atop the Fiasco microkernel and the basic trusted services (L4Env). Each L^{4}Linux instance can communicate with one NIC driver exclusively. Both instances are connected (indirectly) via the trusted Viaduct.

only way of passing data from either side to the other is the Viaduct, which enforces the security policy and protects sensitive data. The scenario is depicted in Figure 2.

Now that we have split the VPN software into distinct components, we can revisit the security measures for each component individually to verify that sensitive information remains protected against unauthorized inspection and manipulation. The Viaduct must be ultimately trusted regarding confidentiality and integrity of the processed data, therefore, its complexity is crucial. Our implementation comprises merely 5,000 LOC plus the used cryptography engine.

The L^{4}Linux instance on the Internet side never observes sensitive information because sensitive data is protected by the Viaduct before leaving the VPN. Therefore we can safely regard this L^{4}Linux instance as untrusted with respect to confidentiality and integrity of sensitive data.

In contrast to the L^{4}Linux instance of the Internet side, the L^{4}Linux instance on the private side observes sensitive data. Nevertheless, network packets cannot leak to the Internet side because of the encapsulation. Thus, we do not need to trust this L^{4}Linux to meet confidentiality. With respect to integrity of the sensitive data, the private L^{4}Linux needs to be trustworthy just as every component of the private network's infrastructure. The key point is that the private L^{4}Linux instance cannot be attacked from the untrusted network because no unauthorized data from the untrusted network passes the Viaduct. In the case of unusual attacks from inside the VPN the VPN Box is vulnerable with respect to integrity. To overcome this, it is necessary to use techniques that shall be described in the next section (Section 3.2).

In summary, the basic architecture of our VPN gateway reduces the trusted computing base of the VPN gateway to the basic Nizza components plus the Viaduct. The overly complex functionality of a complete TCP/IP implementation is provided by untrusted legacy components without compromising our security objectives.

3.2 Availability

Up to now we did not investigate the security objective *availability*. For our scenario, we need to classify attacks against availability. The first class of such attacks aims at producing *overload* to throttle the quality of a service or even make it unusable. Attacks of the second class *crash* a system or components of a system to make it unusable.

To prevent overload attacks, communication protocols and state machines of protocol interpreters need to be designed to be robust against such attacks. For example, the replay window and the sequence number in the IPSec protocol protect the VPN implementation from CPU overload caused by an attacker resending encrypted packets. We have to consider this kind of attacks also in our local communication protocols among our components.

Examining crash attacks, our system seems to be worse than the original monolithic system, because we have twice the complexity of the untrusted L^{4}Linux. However, attacks from the untrusted network can only affect the L^{4}Linux on the public side, because packets that pass the Viaduct have to originate from another trusted VPN island. This means we are on a level equal to a monolithic Linux.

Attacks from within the VPN can only affect the L^{4}Linux on the private side of the VPN gateway, because all traffic that passes the Viaduct to the public network is encrypted and therefore cannot crash the public L^{4}Linux. Nevertheless, attacks from inside the VPN are not typical for VPN scenarios.

With the Nizza architecture, it is even possible to increase the robustness against crash attacks. In contrast to a monolithic system, we are able to reduce all the untrusted components on either side to their bare minimum. In our case we do not need to run any Linux application neither on the public nor on the private side. On the public side we need a complete network-protocol stack that must provide the transport protocols for the key-exchange (IKE) daemon and performs the packet reassembly (see Section 3.3) for the Viaduct. The IKE daemon is a trusted application that manages the exchange of cryptographic keys and protocol negotiation among the VPN gateways of a VPN.

On the private side, we have to consider the topology of the VPN. If the VPN is a single subnet without internal routers, the only functionality besides the NIC device driver is an address resolution protocol (ARP[2]) implementation. The ARP implementation reads the destination IP address from the datagrams that leave the VPN gateway into the private network. However, we need no further functionality for processing IP packets. In the case of a complex VPN topology with internal subnets we need additional functionality for IP routing.

With the resulting architecture as depicted in Figure 3, we minimize the possibilities of denial-of-service attacks while still using complex untrusted software. From the untrusted Internet only the untrusted network-protocol stack can be attacked. Attacks that aim on other facilities of the Linux kernel, such as process management, user management, or file systems, are no longer possible.

[2] ARP is used for translating IP addresses into link-layer addresses (e.g., Ethernet MAC addresses).

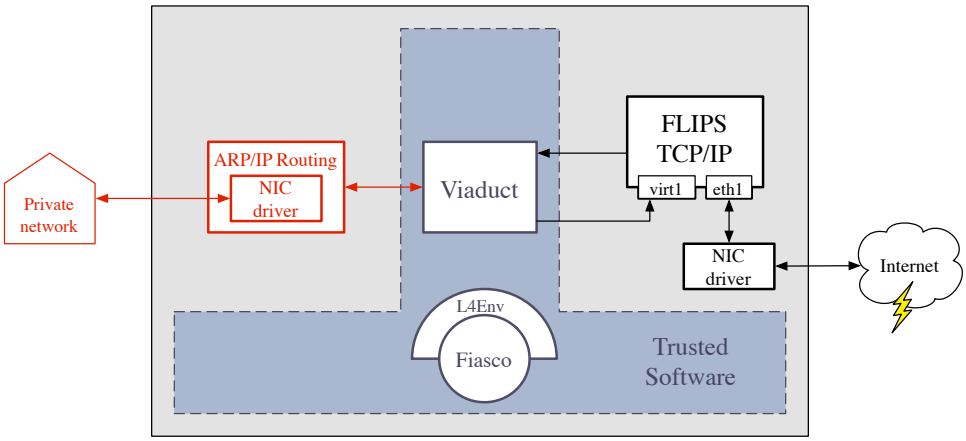

Fig. 3: Minimalized Mikro-SINA architecture. The private L⁴Linux is replaced by a small component that contains only the NIC driver and ARP handling. The public L⁴Linux is replaced by the Flexible TCP/IP Stack (FLIPS). FLIPS is a port of the Linux IP-Stack to the Nizza platform. Its complexity with 80.000 LOC is significantly lower than that of L⁴Linux.

3.3 Dataflow

Next, we describe the data flow in Mikro-SINA in detail (Fig. 2). On each side, a trusted device driver transfers network packets between the physical network and the corresponding L⁴Linux instance. We implemented custom L⁴Linux driver stubs to attach the NIC to its L⁴Linux as proposed in [7].

The private L⁴Linux instance routes network packets arriving at eth0 to a pseudo device driver virt0 attached to the Viaduct and vice versa. In the minimized architecture (see Figure 3) the NIC driver provides the received packets directly to the Viaduct. The packet handling on the Internet side is more complex (Fig. 4). The Viaduct protects outgoing packets via cryptography and forwards them to pseudo device virt1 of the L⁴Linux at the Internet side. L⁴Linux performs further packet processing (e. g., fragmentation) and finally routes the packets to the Internet via eth1.

Incoming packets from the Internet may be fragmented on the way from one VPN gateway to another. These fragments cannot be processed by IPSec individually. The logic to reassemble fragmented IP datagrams is rather complex and does not need to be trustworthy. Therefore, packets must be reassembled before handing them over to the Viaduct. The L⁴Linux TCP/IP implementation performs the packet defragmentation before handing a packet from the IP layer to an upper-layer protocol (e. g., TCP, AH, ESP). Therefore, we use a pseudo IPSec protocol stub in L⁴Linux that forwards the packets to the Viaduct, which applies IPSec functions and sends them to the private L⁴Linux instance.

3.4 Evaluation

In the following, we evaluate our approach in terms of code complexity. We compare Mikro-SINA with monolithic Linux. A typical configuration of the Linux 2.4 kernel comprises about 500,000 LOC (with about 80,000 LOC accounting for the networking

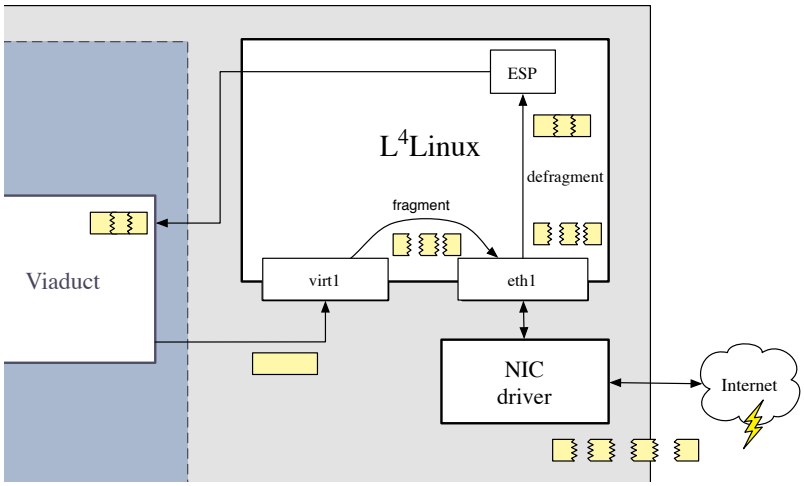

Fig. 4: A closer view of the flow of network packets on the Internet side of the Mikro-SINA VPN gateway. The IP stack of L^{4}Linux reassembles fragmented IPSec packets before they are passed to the higher network layer (pseudo ESP destination). Thus, the Viaduct always receives IPSec packets that are ready to directly apply the needed cryptographic functions.

implementation[3]). For Mikro-SINA, we focused our analysis on the complexity of the components that need to be trusted to meet the hard security requirements—integrity and confidentiality of private data. These components are:

- The microkernel (about 15,000 LOC)
- Basic resource managers (about 30,000 LOC)
- The Viaduct (about 5,000 LOC)

This leads to an overall complexity of merely 50,000 LOC for the TCB, which is an order of magnitude lower than that of the Linux kernel.

4 Related Work

One of the most recent architectures that is comparable to Nizza is the Next-Generation Secure Computing Base (NGSCB) from Microsoft [2, 3].

NGSCB is an operating-system architecture that is quite similar to our Nizza security architecture. Microsoft uses a software architecture akin to our microkernel-based architecture. The microkernel is called Nexus in their publications. NGSCB supports Windows as legacy operating system and provides secure partitions for trusted applications. It is not clear to us whether NGSCB supports scenarios with multiple instances of a legacy operating system. In addition, there is no information on the facilities for cross-domain (cross-partition) communication and their efficiency, which is a precondition for fine-grained isolation of components.

[3]These numbers are for vanilla Linux 2.4.28 without FreeS/WAN patches.

The most important difference between NGSCB and Nizza is that NGSCB describes a special hardware platform [2] that is necessary to run NGSCB software. NGSCB depends on support for the secure applications in all security-critical devices and the infrastructure for that devices, such as the main-board chip set, the video adapter, the keyboard, or the mouse. Most of these hardware changes are necessary because untrusted legacy software has direct access to devices that process unprotected sensitive data.

Nizza is designed to run on standard PC hardware. Nevertheless, for specific security requirements such as attestation of the boot process, we support extended hardware (e. g., a trusted-platform module—TPM). We will also benefit from NGSCB hardware that supports DMA restriction (Section 2.3) or virtualization support, mostly with respect to performance. Because we have small trusted device drivers for the devices that process unprotected sensitive data, we do not need support for protection in these devices.

The aspect of untrusted device drivers is addressed in work by Michael M. Swift and others [11, 12]. They describe how the operating system and applications can be protected from faults in device drivers. This approach uses memory-protection techniques to isolate device drivers from the remaining operating-system kernel. The goal in this work is improved robustness of the operating system while still running unmodified drivers in their usual environment without modifying the interfaces between the drivers and the OS.

With this approach, they reach a fairly good tolerance against programming errors in device drivers and are able to recover from driver crashes. However, they do not aim on hard security objectives that must be maintained even if a faulty device driver or system component becomes malicious.

Another development we have to mention are virtual machines. Virtual-machine techniques became extremely popular in the last years. It seems to be a valuable technique for server consolidation with secure isolation among the different machine instances. With Xen [4], Paul Braham and others propose para-virtualization with slightly modified legacy OS instances as an alternative to complete virtualization. This development is quite similar to our microkernel approach and L^4Linux. Even though there are important differences between pure virtual machines and microkernels, both approaches are promising regarding secure systems. You can find detailed information on that topic in [8].

5 Conclusions

The main goal of this work was minimizing the complexity of the trusted components (in means of LOC) of a VPN gateway while maintaining the functionality of the originally monolithic implementation.

We met this goal by carefully revisiting the software building blocks with respect to the required security measures. We used existing legacy implementations of device drivers and protocols wherever the use of untrusted software is feasible without jeopardizing the system's security. However, untrusted software components must be assumed to

possibly malfunction. While availability cannot be taken for granted, Mikro-SINA protects the confidentiality and integrity of transported information.

The Mikro-SINA approach enabled us to keep our engineering effort, which in essence is our custom IPSec implementation, as small as 5,000 LOC. The overall complexity of the trusted software is only 50,000 LOC.

Mikro-SINA represents a prime example of how to build secure platforms based on the Nizza security architecture and shows the way toward the broad use of Nizza as a base of future security platforms. Our convincing experiences with Mikro-SINA highlight Nizza as an open-source alternative to the upcoming NGSCB platform from Microsoft.

References

[1] Trusted Computing Group website. URL: `https://www.trustedcomputinggroup.org`.

[2] Windows Platform Design Notes: Hardware Platform for the Next-Gerneration Secure Computing Base. Microsoft Corporation, 2003.

[3] Windows Platform Design Notes: Security Model for the Next-Gerneration Secure Computing Base. Microsoft Corporation, 2003.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP)*, pages 164–177, Bolton Landing, NY, October 2003.

[5] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, pages 73–88, Banff, Alberta, Canada, 2001. ACM Press.

[6] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The performance of μ-kernel-based systems. In *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, pages 66–77, Saint-Malo, France, October 1997.

[7] H. Härtig, J. Löser, F. Mehnert, L. Reuther, M. Pohlack, and A. Warg. An I/O Architecture for Microkernel-Based Operating Systems. Technical Report TUD-FI03-08-Juli-2003, Dresden University of Technology, Dresden, Germany, July 2003.

[8] Michael Hohmuth, Michael Peter, Hermann Härtig, and Jonathan S. Shapiro. Reducing TCB size by using untrusted components — small kernels versus virtual-machine monitors. In *Proceedings of the Eleventh ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[9] M. H. Kang and I. S. Moskowitz. A pump for rapid, reliable, secure communication. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 119–129, 1993.

[10] J. Liedtke. Clans & chiefs. In *12. GI/ITG-Fachtagung Architektur von Rechensystemen*, pages 294–305, Kiel, March 1992. Springer.

[11] M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy. Recovering Device Drivers. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, December 2004.

[12] Michael M. Swift, Brian N. Bershard, and Henry M. Levy. Improving the Reliability of Commodity Operating Systems. In *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP)*, pages 207–222, Bolton Landing, NY, October 2003.