Bitfrost: the One Laptop per Child Security Model

Ivan Krstić
One Laptop per Child
Cambridge, MA
krstic@solarsail.hcs.harvard.edu

Simson L. Garfinkel
Naval Postgraduate School
Monterey, CA
simsong@acm.org

ABSTRACT

We present an integrated security model for a low-cost laptop that will be widely deployed throughout the developing world. Implemented on top of Linux operating system, the model is designed to restrict the laptop's software without restricting the laptop's user.

Categories and Subject Descriptors

D.4.6.c [**Security and Privacy Protection**]: Cryptographic Controls; H.5.2.e [**HCI User Interfaces**]: Evaluation/methodology

General Terms

Usability, Security

Keywords

Bitfrost, Linux

1. INTRODUCTION

Within the next year more than five million low-cost laptops will be distributed to children in the developing world who have never before had direct experience with information technology. This "One Laptop per Child" project seeks to use the power of information technology to revolutionize education and learning within the developing world.

The laptops will be sold to countries and educational authorities (hereafter "customers"), not to individuals, and shipped from production facilities at Quanta Computer; further shipping and delivery is the responsibility of the customer. Current plans are for the customers to ship the laptops to schools, where they will be distributed to children.

Each of these children's "XO" laptops will run a greatly slimmed-down version of the Linux operating system and will participate in a wireless mesh network that will connect them to the Internet via gateways located in schools (Figure 2). The laptops will be equipped with web browsers, microphones and cameras so that the students can learn of the world outside their communities and communicate with other children around the world.

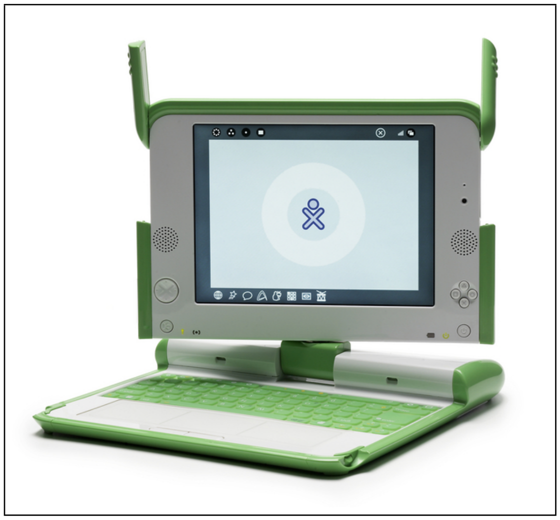


Figure 1: The XO-1 Laptop

Attempting such a project with existing security mechanisms such as anti-virus software and personal firewalls would likely be disastrous: soon after deployment, attackers would inevitably introduce malicious software into the laptop communities. This software might recruit the million-plus laptops to join "botnets." Other attackers might try to disable the laptops out of spite, for sport, as the basis of an extortion scheme, or because they disagree with the project's stated goal of mass education.

Many computer devices that are seen or marketed as "appliances" try to dodge the issue of untrusted or malicious code by only permitting execution of code that is cryptographically signed by the vendor. In practice, this means the user is limited to executing a very restricted set of vendor-provided programs, and cannot develop her own software or use software from third party developers. While this approach certainly limits possible attack vectors, it is not a silver bullet, because even vendor-provided binaries can be exploited—and frequently are.

A more serious problem with the "lockdown" approach is that it would limit what children could do with the laptops that we hope to provide. The OLPC project is based on constructivist learning theories [16]. We believe that by encouraging children to be masters of their computers, they

SOUPS 2007 Pittsburgh, PA

will eventually become masters of their education and develop in a manner that is more open, enthusiastic and creative than they would with a machine that is locked and not "hackable."

As part of our educational mission, we're making it very easy for children to see the code of the programs they're running—we even provide a "View Source" key on the keyboard for this purpose—and are making it similarly easy for children to write their own code in Python, our programming language of choice. Given our further emphasis on collaboration as a feature integrated directly into the operating system, the scenario where a child develops some software and wishes to share it with her friends becomes a natural one, and one that needs to be well-supported.

To this end, we have designed and are implementing Bitfrost, a security platform for the children's laptop that borrows from many recent developments in the field of usable security (HCI-SEC). Freed from the requirement to support legacy software, we believe that we have created a system that may allow children to learn and experiment with advanced technology without falling prey to those who would harm them or their machines.

1.1 Contribution

This paper is the first to present the security model developed for laptops that will be deployed by the One Laptop Per Child project. We also show how the model is implemented with a combination of hardware and software innovations.

As a project that bases most of its software on Linux and other open source offerings, OLPC is yet another project standing on the shoulders of giants. The majority of the security mechanisms and techniques described in this document are no different: most, if not all, have been previously introduced elsewhere. Our contribution is bringing them all together with the unifying vision of creating a laptop with security that can be used by a 5-year-old.

1.2 Outline

This paper is structured as follows: Section 2 gives background information about the OLPC hardware and software. Section 3 lays out our threat model, also outlining areas that are beyond the intended scope for Bitfrost. Section 4 explains our security design goals and principles. Sections 5 explains the physical security portions of Bitfrost: delivery chain security via activation, and anti-theft protection. Section 6 covers how Bitfrost defines and manages user identity, and section 7 details the various Bitfrost protections that apply during normal use of the XO laptop. Section 8 enumerates several security mechanisms missing from the laptops. Section 9 concludes.

2. THE XO ENVIRONMENT

Many of today's production operating systems have lofty goals for usable security but fall short, we believe, because of specific implementation mistakes. Frequently these mistakes are the result of established practices or procedures. After all, every laptop manufactured today must be able to run legacy applications such as Microsoft Word, Firefox, and Emacs.

The XO laptop faces no such restrictions: instead of being designed to allow current computer users to run their existing programs, the XO is built to give millions of children in the developing world their first taste of information technology. Our laptop is thus designed from the ground up for usability and security, low production cost, durability, low power consumption. We accept that any application may need to be modified to run in this environment, although we have worked to make sure that the modifications will be as minimal as necessary. It should also be noted that these modifications are primarily required from GUI applications that deal with the user directly; most software libraries need no modification at all, allowing a vast amount of existing Linux libraries and language bindings to be used normally.

2.1 Hardware

The laptop is based on an AMD Geode LX-700 processor running at 433 MHz. It has a 7.5-inch screen that can operate in either a medium resolution color or high resolution black and white mode, a wireless mesh network, camera that supports video, a microphone, and three USB ports. There is 256 MB of RAM, a 1MB Serial Peripheral Interface (SPI) flash BIOS, and 1GB of high-speed NAND flash storage that holds the laptop's operating system, applications, and user data [12]. Additional storage can be added through the use of external USB devices and the built-in SD card slot.

2.2 Software and Boot Sequence

The laptop's software is a slimmed-down version of Red Hat's Fedora Core Linux distribution. Significant effort has gone to finding and eliminating any program that is unnecessary to the laptop's operation or redundant with other software. The laptop runs the X Window System with the GTK2 graphics toolkit. Window management is provided by Sugar [13], a new user interface approach based on activities, each occurring in their own screen, rather than the traditional interface of overlapping windows, icons, and pull-down menus.

When the laptop is powered up or rebooted, its ENE KB3700 embedded microcontroller reads its firmware out of the laptop's SPI flash. The microcontroller initializes the hardware, then starts the AMD Geode executing the laptop's Open Firmware bootloader [14]. Written largely in Forth, the bootloader can access the mesh network and the laptop's USB ports, navigate the laptop's flash file system, load and execute the Linux kernel, and reflash both the laptop's BIOS and operating system. Under normal circumstances the firmware will load and execute the Linux kernel.

3. SCOPE AND THREAT MODEL

One Laptop per Child is strictly an educational project, using laptops as a means to an end. As a result, in defining the scope of our security platform, we shy away from exotic threats or impractical notions of security. We focus instead on a pragmatic, simple-to-state goal of delivering a machine to children that operates in a predictable way that's consistent with their educational needs, even in the presence of malicious software or attackers.

The threat model for OLPC laptops thus recognizes five core threats. In no particular order, they are: software attacks on the laptop hardware, attacks on operating system integrity, user data loss, attacks on user control of the laptop (and laptop ownership), and attacks on user privacy. Bitfrost aims to mitigate those five classes of threats and no others; we detail these threats below, and briefly mention those explicitly not included in our definition of security.

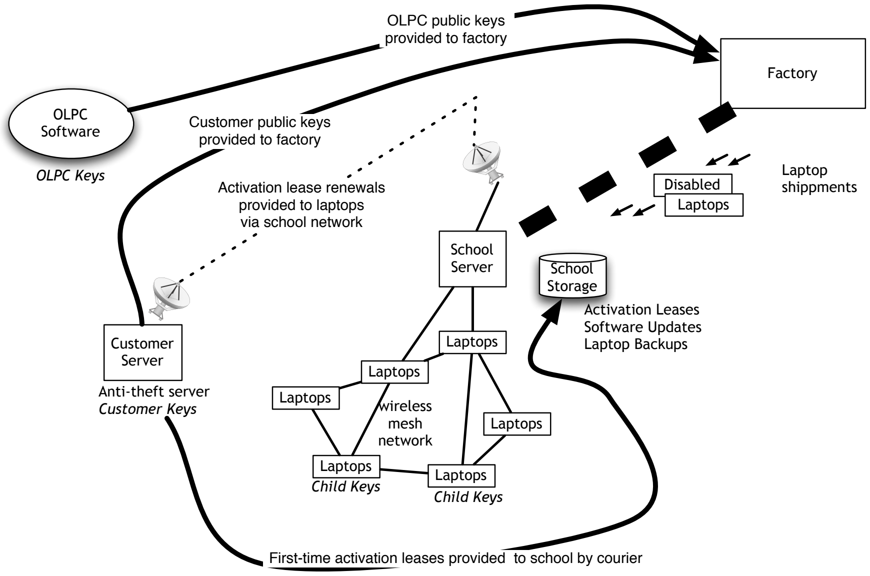


Figure 2: The elements of the One Laptop per Child system discussed in this paper include software provided by OLPC, an anti-theft server run by the laptop customer, the laptop manufacturing plant, the school's server, an Internet connection between the school and the anti-theft system, laptops, and a wireless mesh network that gives the laptops intermittent connectivity to the school and standalone connectivity without user configuration.

3.1 Software attacks on laptop hardware

Due to the unique hardware of the OLPC laptops, malicious software aiming to render a laptop inoperable has two targets at its disposal: the laptop's SPI flash, which contains the BIOS and firmware, and the NAND flash chip, which acts as primary storage in the machine. Overwriting the BIOS or firmware with malicious code or bogus data can either put a machine under unconditional control of a malicious party or prevent it from booting entirely. The NAND flash has a limited number of write/erase cycles before it ceases to function and requires replacement, meaning a piece of malicious code that wears down the chip can cause the laptop to be unusable until trained hardware intervention—in this case, part replacement—is performed.

3.2 Attacks on operating system integrity

Malicious software damaging or deleting parts of the laptop's operating system would, in most cases, be not much more than an annoyance requiring the machine to be reimaged to run. But because we subscribe to constructivist learning theories, we want to strongly encourage children to progress as users, becoming increasingly sophisticated and taking greater liberties with the machine. The possibility of a laptop frequently being rendered inoperable and requiring re-imaging as a result of either hostile software or the child's own experimentation would serve as a strong deterrent to this progression. At the same time, preventing operating system integrity from being subverted under any circumstances runs counter to the idea of giving children full control of their computers and unconstrained freedom to learn and modify what's there.

3.3 User data loss

In the process of encouraging children to understand their computers and master them, it is insufficient to merely ensure that the child can return their machine to a known-good state with regard to operating system integrity in the event of inadvertently causing damage to the OS. Restoring the laptop to a "known-good state" must restore both the OS and the user's documents, pictures, video clips and music, or else children will be strongly discouraged from experimenting with their machines. Similarly, despite the rugged nature of OLPC laptops, it would be upsetting to children if all their documents vanished in case their laptop was damaged or stolen.

3.4 Attacks on user control and ownership

Given the target number of OLPC laptops to be produced in coming years, it could be a profitable endeavor to write malicious software that avoids interfering with a laptop's user or his documents in any way, instead choosing to covertly wrest away control of the laptop by turning it into a spam relay or node in a botnet. In the physical domain, the conspicuous, neon green laptops are easy to spot; given their intended audience, they are presumably just as easy to steal.

3.5 Attacks on user privacy

The presence of a camera and microphone on a laptop intended for very young users is troubling from a privacy point of view, despite their great utility. Malicious software could surreptitiously monitor the children, sending photographs, videos or sound clips to third parties entirely without the child's knowledge or consent. Such software could also violate expectations of the privacy of personal documents and information, which has been shown to have a detrimental impact on the learning process [4].

4. DESIGN GOALS AND PRINCIPLES

After designing our threat model, we created a list of five design goals for OLPC systems security that directly correspond to the five classes of threats we wish to defend against. They are:

Prevent hardware damage The laptop needs to protect itself so that software cannot damage the underlying hardware.

Provide software recoverability It needs to be possible to easily return the laptop to a known-good operating system state using nothing but the laptop itself.

Prevent permanent data loss The information on the laptop must be protected so that it can be recovered in the event that the laptop is lost, stolen or destroyed or the data is deleted or becomes inaccessible for any other reason.

Keep the laptop under control of its owner The laptop should make it difficult for control to be usurped by a third party, both in the software sense by e.g. making the laptop a part of a botnet, and in the physical sense by stealing the laptop.

Protect the user's privacy The laptop needs to be able to prevent information created by the user from being released without the user's explicit permission. Likewise, the laptop's camera or microphone should not be able to covertly monitor the user's actions.

4.1 Implementation Requirements

If there is a set of security principles that inform the design of modern computing systems, it is probably the "anti" principle: if something happens to the computer that the user doesn't like, it's the user's responsibility to obtain, install, configure and use an appropriate piece of anti-software (e.g., anti-virus software, anti-spyware agents, or anti-phishing browsers and tool bars). This reactive approach to security cannot work for OLPC, since we cannot be assured that we will be able to update the deployed laptops with any great speed: were a hostile worm loose on the laptop's mesh network, it is likely that the mesh would deliver the worm to the vulnerable laptops faster than it would deliver the antidote.

For this reason, we developed the OLPC security model while being acutely aware of the kind of experience we wanted the laptop's users to have. This led us past merely defining the security design goals, and towards also defining a set of implementation requirements that govern the user experience with security on the laptops. Namely:

No user passwords With users as young as 5 years old, the security of the laptop cannot depend on the user's ability to remember a password. Users cannot be expected to choose passwords when they first receive computers. As a result, we necessarily treat physical access to the laptop as a proxy for authority to control the laptop. (We considered and rejected the idea of using a graphical authentication system, such as having the child chose a picture to authenticate.)

No reading required Security cannot depend upon the user's ability to read a message from the computer and act in an informed and sensible manner. While disabling a particular security mechanism *may* require reading, a machine must be secure out of the factory if given to a user who cannot yet read.

No lockdown Though in their default settings, the laptop's security systems may impose various prohibitions on the user's actions, there must exist a way for these security systems to be disabled. When that is the case, the machine will grant the user complete control.

Out-of-the-box security The laptop should be both usable and secure out-of-the-box. Security updates should be unnecessary for all but the most critical of vulnerabilities.

Unobtrusive security Whenever possible, the security on the machines must be behind the scenes, making its presence known only through subtle visual or audio cues, and never getting in the user's way. Whenever in conflict with slight user convenience, strong unobtrusive security is to take precedence, though utmost care must be taken to ensure such allowances do not seriously or conspicuously reduce the usability of the machines.

As an example, if a program is found attempting to violate a security setting, the user will not be prompted to permit the action: the action will simply be denied. If the user wishes to grant permission for such an action, she can do so through a special interface we call the Security Center.

4.2 Identification Mechanism

We have seen many security systems fail or become unusable because of their inability to address the identification and authentication of manufacturers, publishers, and users. This is a fundamentally hard problem—one that we cannot hope to solve. Instead, we have taken a very conservative approach to identification management and policy:

Direct certification of content Instead of requiring that the laptop participate in a traditional Public Key Infrastructure, we have decided to provide each laptop with a set of public cryptographic keys that will be used to certify content – such as software or e-books – created by either OLPC or the customer. The sole purpose of these keys will be to verify the integrity of bundled software and content. Note that the purpose of certification is not to restrict execution or viewing to *only* certified content, but to afford such content with additional privileges or capabilities when appropriate.

No unprotected authentication To prevent a wide variety of attacks, we have decided that authentication of laptops or users will never depend upon identifiers that are sent in cleartext over the network. In general, we will refrain from using passwords in the protocols that we develop, and Ethernet MAC addresses will never be used for authentication.

No end-user validation Because of the logistical difficulty, the laptop will not provide services for third-party validation of user identity. Instead, each laptop will associate the public keys that it sees with specific identities and report to users when these associations change, an approach called Key Continuity Management [9].

5. ACTIVATION AND ANTI-THEFT

Because of the production volumes, laptop shipments are an attractive target for theft. What's more, several countries participating in project have high levels of theft and corruption. Not surprisingly, OLPC has received requests to design a system that would act as a strong deterrent against theft.

5.1 Appeal to Children, not Adults

The first element of the anti-theft strategy is a number of engineering choices that were designed to make the laptop unsuitable for running conventional application programs and operating systems and unappealing to adults. For example, the laptop's keyboard is well-suited to children's hands and eyes, but not to larger ones.

5.2 Minimize Resale Value

The anti-theft system's second element is to design the machine to have a minimal resale value. With the exception of the custom display that cannot be used with standard computers, all of the XO's valuable parts are soldered onto the motherboard and thus not easily removable. Even though it has an x86-based processor, the XO lacks the storage and other necessary hardware to run either the DOS or Windows operating systems. The XO's 256MB of RAM cannot be expanded.

5.3 Mandatory Activation

The third element of the anti-theft system is an activation system that renders each laptop unusable from the point of manufacture until it arrives at the intended destination school, and automatically deactivates the laptop if it is reported stolen. We call this system `P_THEFT`.

During production, in each laptop's SPI flash chip is written a random serial number (SN) and a randomly-generated identifier (UUID), and a customer-specific public key K_1. The SN is printed on the laptop's case while the UUID is a secret that is not directly available to user (although privileged programs running on the laptop can use the secret to attest that a program is actually running on a machine). The public key is used by the customer's anti-theft system.

Before the laptop boots, the BIOS looks in the flash file system for a file containing the *laptop activation lease*. This lease contains an expiration date, the laptop's SN, and a signature over the date, the SN, and the laptop's UUID.

- If the lease is not found, the laptop is deemed to be in the *disabled state* and needs to be activated (Section 5.4).
- If the lease is found and the date in the lease has not passed, the laptop is in the *activated state* and boots normally.
- If the lease is found but expired, the laptop is in the *expired state*. The laptop needs a new lease on life to continue working (Section 5.5).

If the laptop is not activated, the firmware will not boot the operating system. Instead, it will wait to be activated.

5.4 Activation

Before a batch of laptops is shipped to each school, the customer uses software provided by OLPC and the private key that is paired with K_1 to generate a set of activation leases for that batch. This *activation list* is loaded onto a USB drive, and delivered to a project handler at the target school separately from the actual laptop shipment. The handler will typically be a teacher or other school administrator. Each activation lease is keyed to a specific laptop, so the activation list sent to one school cannot be used to activate any other laptop batch.

When the USB drive is received, it is plugged into the OLPC-provided school server, or another server running the requisite software that is connected to a wireless access point. School Servers are authenticated through the use of digital certificates which are signed by a key which matches the public key stored in the laptop's BIOS.) Whichever server takes on this role will be called the *School Activation Server*. An activated XO laptop can be used for this purpose, if necessary.

After receiving the matching laptop batch, the school's project handler will be tasked with giving a laptop to each child at the school. When a child turns on the laptop the laptop will securely communicate to the School Activation Server the laptop's SN and a cryptographic hash of the UUID; the server will send back to the laptop its lease, provided that it has one, or return an error if no lease is available. In the event of an error, the laptop will display a message and try again in an hour. (As a fall-back strategy, the laptop can also look for a valid activation lease presented on a file attached to a USB drive.)

The laptop verifies its activation lease by making sure that the SN on the lease matches the SN stored in the BIOS, and verifying the signature on the lease using the public key K_1 in the BIOS. (This key cannot be overwritten without the use of the Developer's Key, discussed in Section 6.3). If the lease is valid the laptop saves this lease in its flash, becomes 'activated,' and proceeds to boot for the first time.

Activation keys by themsevles will not be subject to theft because each activation key can only be used with the specific laptop for which it was made. Activation keys cannot be held for ransom, because the customer can always create more. However, activation keys and laptops can be stolen together. This combination, together with a working school server, could be used to activate the stolen laptops.

The goal of activation is not to eliminate theft in the delivery chain: clearly, the activation list distribution channel is subject to attack even if the list is delivered out of band from the actual laptops themselves. But the presence of the activation list does make large-scale theft significantly more involved, requiring a potential thief to discover details about the activation list distribution and then resort to additional theft or coercion to compromise the system.

5.5 Phone Home

On a regular basis (typically once each day) each laptop will send a message containing the laptop's SN, a timestamp, and a hash of the SN, timestamp and the UUID to the country's anti-theft service. (The message is encrypted with the public key of the anti-theft server.) If the laptop has not been reported stolen, the service will send back to the laptop the current UTC time and a new activation lease. After validating this lease the laptop will reset its clock to the correct UTC time and save the lease in its flash.

The laptop must have the correct time because activation leases expire—if they did not, a thief could simply use a stolen laptop forever, provided that the laptop was never reconnected to the Internet.

Typically leases will have an expiration time of several weeks. In countries with intermittent Internet access leases can be longer—we believe the activation system will be an effective theft deterrent even with 3 month leases. Communities with no Internet connectivity (or that suddenly lose it—for example, as the result of a failed satellite dish) can be sent a USB stick with a new set of leases that can be loaded onto the server or individually loaded onto each laptop. In this manner, even laptops with failed network adapters can continue to operate.

A monotonically increasing counter is used as protection against replay attacks when a lease is being obtained from the server. The real-time clock is protected from tampering in two ways: it is virtualized in the operating system, disallowing unprivileged applications from modifying it. In addition, the firmware will lock the machine and require reactivation if it discovers, during boot, that the machine's clock is set earlier than the timestamp of the last activation lease that was received (with a tolerance of 24 hours). This means removing the RTC battery to reset the clock will not help an attacker.

5.6 Theft

In the event that a laptop is stolen, this information will be reported by the school to the customer's anti-theft service. If a stolen laptop asks the server for a new lease, it will receive instead a signed message telling the laptop to return to the deactivated state. In this case, the laptop must be returned to the school to which it was originally assigned. After the school reports that the laptop is no longer missing, the laptop can be re-activated.

To avoid having a stolen laptop deactivated, a thief might try to use the laptop as a stand-alone computer without network access. In this event, the laptop will cease to function when its current activation lease expires.

The anti-theft system cannot be bypassed as long as software protection system `P_SF_CORE` is enabled (Section 7).

6. IDENTITY MANAGEMENT

The XO laptop includes an Identity Management system whose primary purpose is to prevent students from masquerading as one another in their communications with each other and with teachers. A secondary purpose of the system is to control access to the Internet, which is expected to be a scarce resource. The system is not designed to prevent anonymous communication, to be a deterrent to software theft or other kinds of malicious use, or to provide strong certification of a user's identity for standardized testing, e-commerce, or voting.

As a result of these requirements, we decided against using a traditional identity infrastructure based on certification authorities and certificate chains. Instead, the laptop's identity system is based on self-signed certificates and key continuity management (KCM).

6.1 Child Identity Establishment

After the laptop activates, it automatically runs a program that asks the child for his or her name, takes the child's picture, and generates a public/private key pair. (Entropy from the child's keystrokes and video camera is used to seed the laptop's random number generator.) This key pair is stored (without a pass phrase) in the laptop's flash and is used to sign the child's name and picture. The combination of the name, picture and signature are the child's "digital identity."

The laptop's SN and digital identity are sent to the school's activation server and stored locally in a database. The database is used both for generating class lists and for reporting laptop thefts.

Current plans are for the public components of the digital identity to be serialized using a new certificate format based on the JavaScript Object Notation (JSON) [3], although these certificates could easily be converted to self-signed X.509 certificates if desired.

6.2 Child Identity Use

There will be many opportunities to use the digital identity. For example, in a collaborative education experience, each laptop can display to children a small photograph of the other students that they are working with, much in the same way that programs like AOL Instant Messenger will display chat icons. Teachers can use photographs and digital signatures to match up submitted homework with specific students.

The child's public key can also be used for establishing a virtual private network (VPN) between the laptop and the school server. This can prevent other students from eavesdropping on communications that the student has with an application running at the school and over the Internet—especially important since many of these packets will be routed over a mesh network. Because each key is associated with a student, schools can use the child's public key and the VPN to restrict Internet access to currently enrolled students.

6.3 The Developer's Key

In addition to the child's self-signed identity, each laptop user may optionally request a "Developer's Key." This key is a cryptographic token that is based on OLPC's private key and the laptop's serial number and does not include the child's identity.

The developer's key is not needed for normal use or programming of the laptop. Instead, the developer's key allows the student to modify their laptop in a manner that, if the child is not careful, could render the laptop inoperable. Specifically, the developer's key can be used to change the laptop's kernel, alter the flash allocation strategy, install new operating systems, and so on. Because of this power, the developer's key can also be used to disable the laptop's anti-theft system (Section 5).

OLPC's philosophy is that laptop user should be able to obtain developer's keys but that doing so should take time, because the key poses a risk to the laptop and to the student.

6.4 School Server Authentication

When the laptop is within wireless range of a trusted server (e.g. one provided by OLPC or the customer), the laptop can securely respond to an authentication challenge with its (SN, UUID) tuple. In addition to serving as a means for the school to exercise network access control—some schools will not wish to provide Internet access to alumni, but only to current students—this authentication can unlock extra services like backup and access to a decentralized digital identity system such as OpenID [15].

OpenID is particularly appealing to OLPC, because it can be used to perpetuate passwordless access even on sites that normally require authentication, as long as they support OpenID. The most common mode of operation for current OpenID identity providers is to request password authentication from the user. With an OpenID provider service running on the school server (or other trusted servers), logins to OpenID-enabled sites will simply succeed transparently, because the child's machine has been authenticated in the background using this authentication system.

6.5 Key Continuity Management

Each laptop will maintain a small local database containing each public key that it has encountered and a count of the number of times that the identity has been seen. This information will be subtly conveyed to laptop users—for example, by using colors. This will give the laptop's users a ready means for distinguishing between established identities and newcomers—and for readily identifying malicious interlopers that might try to use another child's name and identity with a new public key.

7. SOFTWARE PROTECTION

The goal of the XO laptop's software protection system is to allow children to execute untrusted code while limiting the ability of this code to inflict harm to the system, on the user, or on other laptops. Whereas most of todays computer security systems are designed with the goal of keeping bad software out of a computer system or network, the XO's security system is designed instead to restrict the functionality allowed to all running programs. There is no reason that a single-user game of Solitaire needs to be able to access the network, read or modify the user's documents or turn on a computer's built-in camera or microphone. On our system this kind of functionality is denied by default to all programs running on the computer, and is only provided to specific applications if the capability is required when the application is installed.

We realize our software protection goal through the use of a non-interference model implemented with cryptographic protection for the BIOS and kernel, protections of the running operating system through the use of the Linux VServer kernel patch [21], and a fine-grain privilege system that requires applications to declare what permissions they need when they are installed.

We believe that most of today's educational and entertainment programs can be written to function within the application limitations that we have designed. However, there are certain to be some applications that cannot function within the limiting polices described below. These applications will be distributed with a second signature, signed with an OLPC or customer key, that tells the installer to disable specific security policies for the application after it is installed.

7.1 The Non-interference Model

The non-interference security model holds that "One group of users, using a certain set of commands, is non-interfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see."[8]

The XO uses a modified version of the noninterference model in which software running on the laptop should be prohibited from interfering with other programs, other laptops, or the laptop's own hardware. We enforce this by preventing software from doing any one of several "bad things:"

Damaging the machine's hardware. Software wishing to render a laptop inoperable may try to ruin the machine's BIOS, preventing it from booting or being reflashed. It may also attempt to exhaust the number of write/erase cycles that the primary NAND chip can perform.

Damaging the machine's software. Software may try to delete or damage the operating system, which would require the machine to be re-imaged and reactivated to run.

Degrading the machine's performance. Software may try to degrade the CPU or drain the battery.

Compromising the user's privacy. We see two threats: software might send user-owned information such as documents and images over the network without authorization; and software might eavesdrop on the user via the laptops' built-in camera or microphone.

Damaging the user's data. Malicious software might attempt to delete or corrupt the user's documents, create large numbers of fake or garbage-filled documents, or attack system services that deal with data, such as the search service.

Damaging the user's reputation by impersonating the user. Malicious software might attempt to abuse the digital identity primitives on the system, such as digital signing, to send messages appearing to come from the user, or to abuse previously authenticated sessions that the user might have created to privileged resources, such as the school server.

Attacking other machines. Malicious software can attack other computers—for example, by launching denial-of-service attacks or by attempting to take them over through the use of binary exploits.

Given the open nature of the XO, we cannot prevent children from intentionally using their laptops to attack others: that type of protection could only be achieved by restricting children to running only a small set of specially pre-approved network applications (and even then the protection might not be complete). Instead, our software security policies are designed to prevent legitimate software from being subverted to malicious ends, and to assure that intentionally malicious software is not installed without the user's express knowledge and permission.

7.2 Core and Runtime Protection

When the laptop is first powered on, the BIOS can either boot a cryptographically signed kernel, refresh the computer's operating system from a signed distribution, or reflash the BIOS from a signed new BIOS. The collection of BIOS protections are collectively referred to as `P_SF_CORE`.

Once the kernel boots, the `P_SF_RUN` runtime protection system takes over. As mentioned above, this system is based on VServer, a lightweight Linux virtualization system that has been widely used at shared hosting Internet service providers for several years. VServer allows us to give different processes different views of the file systems and to significantly restrict the rights afforded to specific processes.

When P_SF_RUN is engaged, the system marks all system files read-only at boot, preventing their modification by any process, including those owned by the superuser.

P_SF_CORE and P_SF_RUN are independent: the system can boot with P_SF_CORE enabled but P_SF_RUN disabled. In this case, VServer creates a copy-on-write (COW) file system that is a union of the underlying operating system and any changes that the user initiates. If P_SF_RUN is later re-enabled, the user's read-only files will include the local modifications. However, these modifications can be reverted by simply throwing away the copy. Thus, the COW system makes it possible for users to make changes to the laptop's operating system and applications while retaining the ability to easily revert these changes without having to reflash the laptop.

7.3 Application Declarations and Installation

As with most computers, we expect that software on the XO will be divided into more or less stand alone applications that communicate with the user through the computer's I/O devices, that may use the network, and that wish to keep persistent information in the computer's file system. But whereas other systems place few if any limitations on applications, the XO runs applications in a restrictive runtime environment.

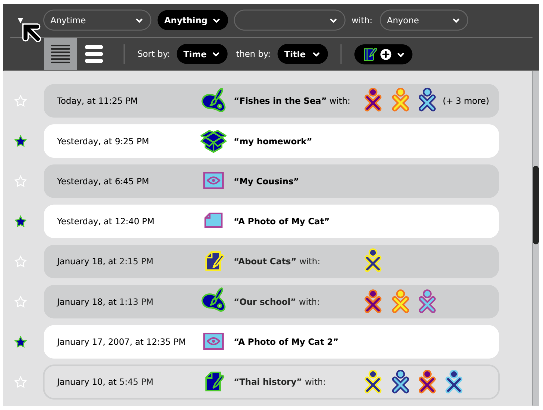
Every application that is installed on the laptop does not need to be able to use every facility that the laptop provides. Yet many security problems in recent years have resulted when attackers have devised some clever way to inject their own code into an existing benign application and force it to do their bidding. Approaches such as StackGuard [2] have attempted to solve this problem by making programs more resistant to code-injection attacks. While we incorporate similar technology on the XO, our fundamental approach to this problem of hostile code is a different one: we limit the damage that injected code can do by limiting each application program to a repertoire of functionality that the application declares when it is installed.

Each application that runs on the XO is distributed as a bundle that includes the executable code, resources, the digital signature of the program's author or publisher, and a list of privileges that the application require. This bundle is read by the laptop's installer service, which queries the bundle for the program's desired security permissions and configures the laptop's Security Service accordingly. After installation, the per-program permission list is only modifiable by the user through a graphical interface.

We plan to use social pressure to convince application developers to distribute their programs with the minimal privileges necessary to accomplish the intended task. We hope that, over time, the laptop's users will become suspicious of programs asking for privileges that seem unnecessary and refuse to install them.

7.4 The Document Journal

Unlike with traditional machines, user documents on the XO laptop are not stored directly on the filesystem. Instead, they are read and stored through a service that provides an object-oriented interface to user documents (see Figure 3). We call this service the Document Journal.



Anytime | Anything | with: Anyone

Sort by: Time | then by: Title

☆ Today, at 11:25 PM "Fishes in the Sea" with: (+ 3 more)

★ Yesterday, at 9:25 PM "my homework"

☆ Yesterday, at 6:45 PM "My Cousins"

★ Yesterday, at 12:40 PM "A Photo of My Cat"

☆ January 18, at 2:15 PM "About Cats" with:

☆ January 18, at 1:13 PM "Our school" with:

★ January 17, 2007, at 12:35 PM "A Photo of My Cat 2"

☆ January 10, at 5:45 PM "Thai history" with:

Figure 3: The Document Journal

Programs on the XO may not use the `open()` call to arbitrarily open user documents in the system, nor can they use any version of the `opendir()` or `readdir()` calls to list available documents. Instead, when a program wishes to open a user document, it asks the system to present the user with a 'file open' dialog. If the user selects a file, the open file descriptor is returned to the program. This system provides protection for data in the journal against malicious programs; we call this protection system (P_DOCUMENT).

Benign programs are not adversely impacted by the need to use the file store for document access, because they generally do not care about rendering their own file open dialogs (the rare exception is programs that create custom dialogs to e.g. offer built-in file previews; for the time being, we are not going to support this capability).

Malicious programs, however, lose a tremendous amount of ability to violate the user's privacy or damage her data, because all document access requires explicit assent by the user.

7.4.1 *Journal Rate Limiting* (P_DOCUMENT_RL)

The file store is rate limited so that programs may not store new files or new versions of old files faster than a predetermined rate (e.g., once every 30 seconds).

7.4.2 *Journal Browsing* (P_DOCUMENT_RO)

Certain kinds of software, such as photo viewing programs, need access to all documents of a certain kind (e.g. images) to fulfill their desired function. This is in direct opposition with the P_DOCUMENT protection which requires user consent for each document being opened.

To resolve the quandary, we must ask ourselves: "from what are we trying to protect the user?." The answer, here, is a malicious program which requests permission to read all images, or all text files, or all e-mails, and then sends those documents over the network to an attacker or posts them publicly, seriously breaching the user's privacy.

We solve this by allowing programs to request read-only permissions for one type of document (e.g. image, audio, text, e-mail) at installation time, but making that permission (P_DOCUMENT_RO) mutually exclusive with asking for any network access at all. A photo browsing program would therefore be unable to connect to the Internet without the user's explicit permission.

7.4.3 *Journal Backup* (P_DOCUMENT_BACKUP)

The laptop will automatically perform incremental backups of user documents whenever it is in range of servers that advertise themselves as offering a backup service.

Because we wish to avoid having children generate a new digital identity if their laptop is ever lost, stolen or broken, by default the child's cryptographic keypair is also backed up to the primary backup server (*i.e.*: the server operated by the school). Given that a child's private key normally has no password protection, stealing the primary backup server (normally the school server) offers the thief the ability to impersonate any child in the system. For now, we deem this an acceptable risk, and we assume that school servers will be provided with an adequate level of physical and logical security.

7.5 NAND Protection Policies

As discussed above, the laptop's NAND flash can only perform a limited number of erase/rewrite cycles before it becomes unusable. Under normal usage this lifetime is not a consideration, as the laptop's JFFS2 [22] filesystem moves stored files around as necessary to assure even leveling of the entire device. However, an unchecked malicious program that continually wrote random, uncompressable data could use up the flash device's lifetime within a few days, whereas normally we expect the flash to last the laptop's lifetime.

7.5.1 *Protecting the NAND flash (*P_NAND_RL*)*

To prevent malicious software from prematurely aging the flash, each program's ability to issue write commands is controlled by a token bucket (similar token buckets are used for network flow control; see for example [17]). When the bucket is drained writes will either be delayed with an exponential backoff or blocked (depending on the result of field testing.)

7.5.2 *Sandbox and Scratch Space* (P_SANDBOX*)*

Each XO application executes in a fortified `chroot()`'ed filesystem that is severely restricted. It normally has no access to system paths such as /proc or /sys, cannot see other programs on the system or their scratch spaces, and only the libraries it needs are mapped into its scratch space. It cannot access user documents directly, but only through the document journal service (Section 7.4).

Every program scratch space has three writable directories, called 'tmp', 'conf', and 'data'. The program is free to use these for temporary, configuration, and data (resource) files, respectively. The rest of the scratch space is immutable; the program may not modify its binaries or core resource files. This model ensures that a program may be restored to its base installation state by emptying the contents of the three writable directories, and that it can be completely uninstalled by removing its bundle (scratch space) directory.

To prevent disk exhaustion attacks, each program is given a maximum of 5MB of NAND space where the application can store configuration information and temporary files.

7.6 Privacy Protection

The XO's journal facility (Section 7.4) prevents software from browsing the user's files by requiring that each file be explicitly opened by the user. The journal browsing limitation (P_DOCUMENT_RO) protects the user from wide-scale privacy breaches by software that purports to be a "viewer" of some broad class of documents.

In addition to these protections, the laptop supports several other protection policies.

7.6.1 *Microphone and Camera Protection* (P_MIC_CAM)

We will have two LEDs, one each for the camera and microphone, which will be lit by the hardware whenever the corresponding device is engaged. The LEDs turning on unexpectedly should tip off the user to potential eavesdropping. Having the LEDs blink for this purpose was evaluated, but was thought too distracting; the LEDs are always on when active. Care was also taken to avoid hysteresis attacks: it is not possible to enable capture from the microphone or camera for such a short time that the LEDs do not noticeably light.

The use of the camera and microphone requires a special permission which must be requested at install-time for each program wishing to do so. This permission does not allow a program to instantly turn on the camera and microphone. Instead, it merely lets the program *ask* the user to allow the camera or microphone (or both) to be turned on.

Thus, any benign programs which are taken over but haven't declared themselves as needing the camera or microphone cannot be used to turn on either.

Programs which have declared themselves as requiring those privileges (for example, a videoconferencing app) can instruct the system to ask the user for permission to enable the camera and microphone components. If the request is granted, the program is granted the capability to toggle the components to the "on" state for a limited period of time (e.g. 30 minutes).

Programs cryptographically signed by a trusted authority will be exempt from having to ask permission to manipulate the components, but because of the LEDs which indicate their status, the potential for abuse is still rather low.

7.6.2 *Background Sound Permission* (P_DSP_BG)

Programs with this permission may play audio when running in the background. This permission prevents otherwise benign programs from being taken over and used to play annoying or embarrassing sounds.

7.6.3 *X Window System Protection* (P_X)

When manually assigned to a program by the user, this permission lets a program send synthetic mouse X events to another program. Its purpose is to enable the use of accessibility software such as an on-screen keyboard. The permission is NOT requestable at install-time, and thus must be manually assigned by the user through a graphical interface, unless the software wishing to use it is cryptographically signed by a trusted authority.

Without this permission, programs cannot eavesdrop on or fake one another's events, which disables key logging software or sophisticated synthetic event manipulation attacks, where malicious software acts as a remote control for some other running program.

7.7 Other Protection Policies

7.7.1 *Network Protection (*P_NET*)*

Applications must specially declare that they wish to make use of the network to either receive or initiate connections over the wireless mesh. We are considering the following kinds of network limitations:

- Boolean network on/off restriction
- Bandwidth throttling with burst allowance
- Connection rate limiting
- Packet destination restrictions by host name, IP and port(s)
- Time-of-day restrictions on network use
- Data transfer limit by hour or day
- Server restrictions: Boolean and per-port

Reasonable default rate and transfer limits will be imposed on all programs except those explicitly granted unrestricted network access by OLPC or the customer. Additional restrictions might be added to this list as we complete our evaluation of network policy requirements.

7.7.2 *CPU rate limiting* (P_CPU_RL)

Foreground programs may use all of the machine's CPU power. Background programs will be limited to a fixed amount (currently 10%) unless given explicit permission by the user in the Security Center.

The XO's UI environment (currently named "Sugar") only supports maximized application windows, not the overlapping windows popular on most computers today. When we talk about foreground and background execution, we are referring to programs that are, or are not, currently in control of the entire screen.

7.7.3 *Real Time Clock Protection* (P_RTC)

Because the computer's real time clock is used by the anti-theft system, we cannot allow the user to set or change the clock. This creates a problem for some applications which require the need to make adjustments—for example, we have a music program that must synchronize to within 10ms with any machines with which it co-plays a tune.

To overcome this difficulty, each running application has its own real time clock offset which it is free to change. The offset is lost when the program exits.

8. WHAT'S NOT ON THE LIST

Several features traditionally associated with laptops or computing devices for children are missing from our list. Our reasons for not including these features are practical, not ideological.

8.1 Filesystem encryption

Cryptographic file systems are increasingly popular on portable devices because cryptography can protect information even if an adversary has physical possession of the device—provided that the encryption key is not physically present on the captured device, of course.

We have not implemented a cryptographic file system on the XO for two reasons. First, because we intend the laptop to be used by very young users, we have explicitly designed the laptop so that it does not rely on passwords or other user-remembered secrets. Of course, without a password or some kind of physical token that is separate from the laptop, there is no way to protect the cryptographic key in the event that the laptop is stolen—thus, the cryptographic file system would not provide any real protection.

The second reason that we had when originally deciding against implementing a cryptographic file system is that the laptop's original CPU was not up to the task. The XO-1 was targeted to ship with the AMD Geode GX-500 processor running at 366 MHz, which could only encrypt about 4MB/s with the AES-128 algorithm in CBC mode, a tenth of the throughput of the NAND flash chip. However, it was decided recently that the laptops will ship with a Geode LX-700 which features an on-board AES accelerator. This makes it very likely that we will offer the ability of filesystem encryption in the future, by way of a P_PASSWORD protection that might be enabled on laptops used by older children. In the meantime, we expect tools will be made quickly available by the community that allow children to encrypt individual files or directories, e.g. by providing graphical front-ends to available tools such as GPG.

8.2 Objectionable content filtering

As outlined in the threat model, the Bitfrost platform governs system security on the XO laptops, but does not have any provisions for content control. Customers who desire filtering of material of sexual, political or religious nature will almost certainly implement these filtering policies at the schools using special-purpose software or at the national level.

8.3 Multiple Users

Although our current approach is to design the laptop for use by a single child, it is quite likely that some laptops will be used by other children or family members. At some point in the future we may provide for multiple users, but at the present this is beyond our engineering capacity.

It should be noted that no security mechanisms described in this document require modification to support multiple users. Instead, the required changes lie in the general OS and GUI layers

9. CONCLUSION

Bitfrost takes is name from Bifröst, the mythical Norse rainbow bridge which keeps mortals, inhabitants of the realm of Midgard, from venturing into Asgard, the realm of the gods. In effect, Bifröst is a majestic security system designed to keep out unwanted intruders. In the 12th century Icelandic historian and poet Snorri Sturluson wrote of the bridge, "It is of three colors, and very strong, and made with cunning and with more magic art than other works of craftsmanship. But strong as it is, yet must it be broken, when the sons of Múspell shall go forth harrying and ride it, and swim their horses over great rivers; thus they shall proceed. . . nothing in this world is of such nature that it may be relied on when the sons of Múspell go a-harrying."[20]

This story is quite remarkable, as it amounts to a 13th century recognition of the idea that there's no such thing as a perfect security system.

To borrow Sturluson's terms, we believe that we have imbued the OLPC security system with cunning and more magic art than other similar works of craftsmanship—but not for a second do we think that we have designed something that cannot be broken when talented, determined and resourceful attackers go forth harrying. This was not the goal. The goal was to significantly raise the bar from the current, deeply unsatisfactory, state of desktop security. We believe Bitfrost accomplishes this, though only once the laptops are deployed in the field will we be able to tell with some degree of certainty whether we have succeeded.

9.1 Related Work

In 1982 Goguen and Meseguer at SRI proposed that the principle of noninterference could be used as a model for computer security in multi-user operating systems [8]. The noninterference model was a reaction to the difficulty that the industry had experienced in implementing the Bell-Lapadula security model, and the realization that the model did not protect real-world computer systems against many real-world problems [1].

Reid observed in 1987 that users rarely need to run software that has not been properly installed [18]. Kirovski, Drinic and Potkonjak made the same observation in 2002 and developed a working implementation that limited execution to installed software. [11]

The idea of a user-auditable application manifest describing particularly notable application functions was proposed by Garfinkel as part of software "label" mandated by a hypothetical "Pure Software Act" [5], although no enforcement mechanism was imposed beyond regulation and litigation.

The "security by designation" approach used by the XO's journal is directly based upon the work of Yee [24], while the isolation provided between XO applications is similar to the Polaris system developed by Stiegler *et. al.* [19]. It achieves a compromise similar to what the XO operating system is after: enforcing privilege separation while maintaining compatibility with applications written for ambient authority operating systems. The underlying philosophy of secure interaction design is inspired by Yee as well [23].

The laptop's bottom-up PKI, known as Key Continuity Management, was proposed as a security model by Gutmann [9] and analyzed by Garfinkel [6]. Garfinkel conducted an analysis of KCM's strengths and weaknesses against a variety of attacks [7].

Techniques for restricting root in open source operating systems were pioneered by the FreeBSD "jail" facility [10] and refined by the VServer project [21].

Product activation and "phone home" schemes are widely used in today's software industry.

9.2 Acknowledgments

George Dinolt at NPS and Ka-Ping Yee at Berkeley provided useful feedback on an earlier draft of this article.

10. REFERENCES

[1] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. report MTR 2547 v2. Technical report, MITRE, November 1973.

[2] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proc. 7th USENIX Security Conference*, pages 63–78. Usenix, San Antonio, Texas, jan 1998. `citeseer.ist.psu.edu/cowan98stackguard.html`.

[3] D. Crockford. RFC 4627: The application/json media type for javascript object notation (json), July 2006.

[4] Peter Elbow. In defense of private writing. *Written Communication*, 16(2):139–170, 1999.

[5] Simson Garfinkel. The pure software act of 2006. *TechnologyReview.com*, April 7 2004. `http://simson.net/clips/2004/2004.TR.04.PureSoftware.pdf`.

[6] Simson L. Garfinkel. *Design Principles and Patterns for Computer Systems that are Simultaneously Secure and Usable*. PhD thesis, MIT, Cambridge, MA, April 26 2005.

[7] Simson L. Garfinkel and Robert Miller. The johnny 2 standardized secure messaging scenario. In *Symposium on Usable Privacy and Security*. ACM Press, 2005.

[8] Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proceedings of the Berkeley Conference on Computer Security*, pages 11–22. IEEE CS Press, 1982.

[9] Peter Gutmann. Why isn't the Internet secure yet, dammit. In *AusCERT Asia Pacific Information Technology Security Conference 2004; Computer Security: Are we there yet?* AusCERT, May 2004. `http://www.cs.auckland.ac.nz/~pgut001/pubs/dammit.pdf`.

[10] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *System Administration and Network Engineering (SANE) 2000*. Stichting NLnet and USENIX, May 2000. `http://docs.freebsd.org/44doc/papers/jail/jail.html`.

[11] D. Kirovski, M. Drinic, and M. Potkonjak. Enabling trusted software integrity. In *In Proceedings of ASPLOS*, pages 108–120, 2002.

[12] OLPC. Hardware specification, 2007. `http://wiki.laptop.org/go/Hardware_specification`.

[13] OLPC. Sugar, 2007. `http://wiki.laptop.org/go/Sugar`.

[14] Open firmware home page, 2007. `http://www.openfirmware.org`.

[15] What is OpenID, 2007. `http://openid.net`.

[16] Seymour Papert and Idit Harel. *Constructionism*. Ablex Publishing Corporation, 1991.

[17] C. Partridge. RFC 1363: A proposed flow specification, September 1992. Status: INFORMATIONAL.

[18] Brian Reid. Reflections on some recent widespread computer break-ins. *Commun. ACM*, 30(2):103–105, 1987. ISSN 0001-0782.

[19] Marc Stiegler, Alan H. Karp, Ka-Ping Yee, Tyler Close, and Mark S. Miller. Polaris: virus-safe computing for windows xp. *Commun. ACM*, 49(9): 83–88, 2006. ISSN 0001-0782.

[20] Snorri Sturluson. *Edda*. Everyman Paperback Classics, 1995.

[21] Linux VServer. `http://linux-vserver.org/`.

[22] David Woodhouse. The journaling flash file system, July 2001.

[23] Ka-Ping Yee. User interaction design for secure systems. In *Proceedings of the 4th International Conference on Information and Communications Security*. Springer-Verlag, 2002. LNCS 2513.

[24] Ka-Ping Yee. Aligning security and usability. *Security & Privacy Magazine*, 2:48–55, Sept–Oct 2004.