One Laptop per Child Security Specification

Contents

A. Introduction 3

1. Foreword
2. Security and OLPC
3 About this document
4 Principles and goals
    a. Principles
    b. Goals

B. Factory production 8

C. Delivery chain security 9

D. Arrival at school site and activation 10

E. First boot 11

F. Software installation 12

G. Software execution: problem statement 14

H. Threat model: bad things that software can do 15

1. Damaging the machine
2. Compromising privacy
3. Damaging the user's data
4. Doing bad things to other people
5. Impersonating the user

I. Protections 18

1. P_BIOS_CORE: core BIOS protection
2. P_BIOS_COPY: secondary BIOS protection
3. P_SF_CORE: core system file protection
4. P_SF_RUN: running system file protection
5. P_NET: network policy protection
6. P_NAND_RL: NAND write/erase protection

7. P_NAND_QUOTA: NAND quota
8. P_MIC_CAM: microphone and camera protection
9. P_CPU_RL: CPU rate limiting
10. P_RTC: real time clock protection
11. P_DSP_BG: background sound permission
12. P_X: X window system protection
13. P_IDENT: identity service
14. P_SANDBOX: program jails
15. P_DOCUMENT: file store service
16. P_DOCUMENT_RO
17. P_DOCUMENT_RL: file store rate limiting
18. P_DOCUMENT_BACKUP: file store backup service
19. P_THEFT: anti-theft protection
20. P_SERVER_AUTH: transparent strong authentication to trusted server
21. (For later implementation) P_PASSWORD: password protection

J. Addressing the threat model 27

1. Damaging the machine
2. Compromising privacy
3. Damaging the user's data
4. Doing bad things to other people
5. Impersonating the user
6. Miscellaneous
7. Missing from this list
    a. Filesystem encryption
    b. Objectionable content filtering

K. Laptop disposal and transfer security 30

L. Closing words 31

A. Introduction

1. Foreword

In 1971, 35 years ago, AT&T programmers Ken Thompson and Dennis Ritchie released the first version of UNIX. The operating system, which started in 1969 as an unpaid project called UNICS, got a name change and some official funding by Bell Labs when the programmers offered to add text processing support. Many of the big design ideas behind UNIX persist to this day: popular server operating systems like Linux, FreeBSD, and a host of others all share much of the basic UNIX design.

The 1971 version of UNIX supported the following security permissions on user files:

- Non-owner can change file (write)
- Non-owner can read file
- Owner can change file (write)
- Owner can read file
- File can be executed
- File is set-uid

These permissions should look familiar, because they are very close to the same security permissions a user can set for her files today, in her operating system of choice. What's deeply troubling—almost unbelievable—about these permissions is that they've remained virtually the *only* real control mechanism that a user has over her personal documents today: a user can choose to protect her files from other people on the system, but has no control whatsoever over what her own programs are able to do with her files.

In 1971, this might have been acceptable: it was 20 years before the advent of the Web, and the threat model for most computer users was entirely different than the one that applies today. But how, then, is it a surprise that we can't stop viruses and malware now, when our defenses have remained largely unchanged from thirty-five years ago?

The crux of the problem lies in the assumption that any program executing on a system on the user's behalf should have the exact same abilities and permissions as any other program executing on behalf of the same user. 1971 was seven years before the first ever international packet-switched network came into existence. And the first wide-area network using TCP/IP, the communication suite used by the modern Internet, wasn't created until 1983, twelve years after Thompson and Ritchie designed the file permissions we're discussing. The bottom line is that in 1971, there was almost no conceivable way a program could "come to exist" on a computer except if the account owner—the user—physically transported it to a machine (for instance, on punched tape), or entered it there

manually. And so the "all or nothing" security approach, where executing programs have full control over their owner's account, made quite a lot of sense: any code the user executed, she ipso facto trusted for all practical purposes.

Fast forward to today, and the situation couldn't be more different: the starkest contrast is perhaps the Web, where a user's web browser executes untrusted scripting code on just about every web page she visits! Browsers are growing increasingly complex sandboxing systems that try to restrict the abilities of such web scripts, but even the latest browser versions are still fixing bugs in their scripting engine implementations. And don't forget e-mail: anyone can send a user an executable program, and for many years the users' instinctive reaction was to open the attachment and run the program. Untrusted code is everywhere, and the only defense seems to be tedious user training and antivirus software—the latter assuming it's fully updated, and assuming the antivirus makers have had time to deconstruct each latest virus and construct a defense for it.

Most technologies and approaches mentioned in the rest of this document do not represent original research: they have been known in the security literature for years, some of them have been deployed in the field, and others are being tested in the lab. What makes the OLPC XO laptops radically different is that they represent the first time that all these security measures have been carefully put together on a system slated to be introduced to tens or hundreds of millions of users. The laptops are also possibly the first time that a mainstream computing product has been willing to give up compatibility with legacy programs in order to achieve strong security. As an example, you'll find that talk about anti-virus and anti-spyware technology is conspicuously absent from this document, because the Bitfrost security platform on the XO laptops largely renders these issues moot.

We have set out to create a system that is both drastically more secure and provides drastically more usable security than any mainstream system currently on the market. One result of the dedication to usability is that there is only one protection provided by the Bitfrost platform that requires user response, and even then, it's a simple 'yes or no' question understandable even by young children. The remainder of the security is provided behind the scenes. But pushing the envelope on both security and usability is a tall order, and as we state in the concluding chapter of this document, we have neither tried to create, nor do we believe we have created, a "perfectly secure" system. Notions of perfect security are foolish, and we distance ourselves up front from any such claims.

2. Security and OLPC

In terms of security, the OLPC XO laptops are a highly unique environment. They are slated to introduce computers to young children, many in environments that have had no prior exposure to computing or the Internet.

What's more, OLPC is not targeting small-scale local deployments where it could easily intervene in the case of security problems with the machines or their usage; instead, once the machines are released in the wild, drastic changes in the security model should be considered difficult or impossible.

Plenty of experience exists in locking down user machines, often in corporate or academic settings. But OLPC has a final constraint that invalidates most of the current common wisdom: OLPC is, by design, striving to be an eminently malleable platform, allowing the children to modify, customize, or "hack", their own machines any way they see fit.

As a result, no one security policy on the computer will satisfy our requirements. Instead, we will ship and enable by default a stringent policy that's appropriate even for the youngest user, and which delivers the strongest available protections. However, we will provide a simple graphical interface for interested users to disable any of these protections, allowing the user to tailor the security level to match her interest in hacking her machine.

This approach allows us to be highly secure by default, and protect even the user who has no conception of digital security. At the same time, it avoids getting in the way of any user who is becoming more sophisticated, and interested in increasing her abilities on the machine.

Finally, because we subscribe to constructionist learning theories, we want to encourage children to all eventually progress to this level of a more sophisticated user who takes greater liberties with her machine. However, as long as there exists potential for disaster (i.e. rendering a machine fully inoperable, or incurring total data loss), this potential serves as a strong deterrent to this progression. Because of this, in addition to focusing on security by default, we are explicitly focusing on providing mechanisms for trivial and unintimidating disaster recovery, such as operating system recovery from multiple sources and data backup to a central server.

3. About this document

This document follows security throughout the life-cycle of the laptop itself, starting from the moment a laptop is produced in the factory, to the moment it first reaches a child, throughout the child's use of the laptop, and finally stopping at the moment a child wishes to dispose of the laptop. All of this is preceded by a

short section on our goals and principles, which serves to provide background to some of the decisions we made, and which might be non-obvious if one thinks of security in the context of normal laptop and desktop machines.

This document is complete with regard to the OLPC security model, but is generally non-technical. A separate document is being prepared that complements this one with fully technical descriptions and commentary.

4. Principles and goals

a. Principles

Open design

The laptop's security must not depend upon a secret design implemented in hardware or software.

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.

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.

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 the graphical security center interface.

b. Goals

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.

No unencrypted authentication

Authentication of laptops or users will not depend upon identifiers that are sent unencrypted over the network. This means no cleartext passwords of any kind will be used in any OLPC protocol and Ethernet MAC addresses will never be used for authentication.

Out-of-the-box security

The laptop should be both usable and secure out-of-the-box, without the need to download security updates when at all possible.

Limited institutional PKI

The laptop will be supplied with public keys from OLPC and the country or regional authority (e.g. the ministry or department of education), but these keys will not be used to validate the identity of laptop users. The sole purpose of these keys will be to verify the integrity of bundled software and content. Users will be identified through an organically-grown PKI without a certified chain of trust—in other words, our approach to PKI is KCM, or key continuity management.

No permanent data loss

Information on the laptop will be replicated to some centralized storage place so that the student can recover it in the even that the laptop is lost, stolen or destroyed.

B. Factory production

As part of factory production, certain manufacturing data is written to the built-in SPI flash chip. The chip is rewritable, but barring hardware tampering, only by a trusted process that will not damage or modify the manufacturing information.

Manufacturing data includes two unique identifiers: SN, the serial number, and U#, the randomly-generated UUID. Serial numbers are not assigned in order; instead, they are chosen randomly from a pool of integers. The manufacturing process maintains a mapping of the random serial number assigned, to the real, incremental serial number which was set to 1 for the first laptop produced. This mapping is confidential but not secret, and is kept by OLPC.

The random mapping's sole purpose is to discourage attempts at using serial numbers of laptops delivered to different countries for attempting to analyze countries' purchase volumes.

A laptop's UUID, U#, is a random 32-byte printable ASCII identifier.

In one of the factory diagnostics stages after each laptop's production, the diagnostics tool will send the complete manufacturing information, including U#, SN, and factory information, to an OLPC server. This information will be queued at the factory in case of connectivity issues, and so won't be lost under any foreseeable circumstances.

At the end of the production line, the laptop is in the 'deactivated' state. This means it must undergo a cryptographic activation process when powered on, before it can be used by an end user.

C. Delivery chain security

OLPC arranges only the shipment of laptops from their origin factory to each purchasing country. Shipping and delivery within each country is organized fully by the country.

Given OLPC production volumes, the delivery chain poses an attractive attack vector for an enterprising thief. The activation requirement makes delivery theft highly unappealing, requiring hardware intervention to disable on each stolen laptop before resale. We give an overview of the activation process below.

D. Arrival at school site and activation

Before a batch of laptops is shipped to each school, the country uses OLPC-provided software to generate a batch of activation codes. This "activation list" maps each (SN, UUID) tuple to a unique activation code for the referenced laptop. Activation lists are generated on-demand by the country for each laptop batch, as the laptops are partitioned into batches destined for specific schools. In other words, there is no master activation list.

The activation list for a laptop batch is loaded onto a USB drive, and delivered to a project handler in the target school out of band from the actual laptop shipment. The handler will be commonly a teacher or other school administrator. The activation list sent to one school cannot be used to activate any other laptop batch.

When the activation list 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. Whichever server takes on this role will be called the '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 receives a laptop, it is still disabled. The child must power on the laptop within wireless range of the school's activation server. When this happens, the laptop will securely communicate its (SN, UUID) tuple to the server, which will return the activation code for the laptop in question, provided the tuple is found in the activation list, or an error if it isn't.

Given an invalid activation code or an error, the laptop will sleep for one hour before retrying activation. If the activation code is valid, the laptop becomes “activated,” and proceeds to boot to the first-boot screen. A textual activation code can be entered into the machine manually, if the machine is not activating automatically for any reason.

E. First boot

On first boot, a program is run that asks the child for their name, takes their picture, and in the background generates an EEC key pair. The key pair is initially not protected by a passphrase, and is then used to sign the child's name and picture. This information and the signature are the child's 'digital identity'.

The laptop transmits the (SN, UUID, digital identity) tuple to the activation server. The mapping between a laptop and the user's identity is maintained by the country or regional authority for anti-theft purposes, but never reaches OLPC.

After this, the laptop boots normally, with all security settings enabled.

F. Software installation

There is a very important distinction between two broad classes of programs that execute on a running system, and this distinction is not often mentioned in security literature. There are programs that are purposely malicious, which is to say that they were written with ill intent from the start, such as with viruses and worms, and there are programs which are circumstantially malicious but otherwise benign, such as legitimate programs that have been exploited by an attacker while they're running, and are now being instrumented to execute code on behalf of the attacker via code injection or some other method.

This difference is crucial and cannot be understated, because it's a reasonable assumption that most software running on a normal machine starts benign. In fact, we observe that it is through exploitation of benign software that most malicious software is first *introduced* to many machines, so protecting benign software becomes a doubly worthy goal.

The protection of benign software is a keystone of our security model. We approach it with the following idea in mind: benign software will not lie about its purpose during installation.

To provide an example, consider the Solitaire game shipped with most versions of Microsoft Windows. This program needs:

- no network access whatsoever
- no ability to read the user's documents
- no ability to utilize the built-in camera or microphone
- no ability to look at, or modify, other programs

Yet if somehow compromised by an attacker, Solitaire is free to do whatever the attacker wishes, including:

- read, corrupt or delete the user's documents, spreadsheets, music, photos and any other files
- eavesdrop on the user via the camera or microphone
- replace the user's wallpaper
- access the user's website passwords
- infect other programs on the hard drive with a virus
- download files to the user's machine
- receive or send e-mail on behalf of the user
- play loud or embarrassing sounds on the speakers

The critical observation here is not that Solitaire should never have the ability to do any of the above (which it clearly shouldn't), but that its creators *know* it

should never do any of the above. It follows that if the system implemented a facility for Solitaire to indicate this at installation time, Solitaire could irreversibly shed various privileges the moment it's installed, which severely limits or simply destroys its usefulness to an attacker were it taken over.

The OLPC XO laptops provide just such a facility. Program installation does not occur through the simple execution of the installer, which is yet another program, but through a system installation service which knows how to install XO program bundles. During installation, the installer service will query the bundle for the program's desired security permissions, and will notify the system Security Service accordingly. After installation, the per-program permission list is only modifiable by the user through a graphical interface.

A benign program such as Solitaire would simply not request any special permissions during installation, and if taken over, would not be able to perform anything particularly damaging, such as the actions from the above list.

It must be noted here that this system *only* protects benign software. The problem still remains of intentionally malicious software, which might request all available permissions during installation in order to abuse them arbitrarily when run. We address this by making certain initially-requestable permissions mutually exclusive, in effect making it difficult for malicious software to request a set of permissions that easily allow malicious action. Details on this mechanism are provided later in this document.

As a final note, programs cryptographically signed by OLPC or the individual countries may bypass the permission request limits, and request any permissions they wish at installation time.

G. Software execution: problem statement

The threat model that we are trying to address while the machine is running normally is a difficult one: we wish to have the ability to execute generally untrusted code, while severely limiting its ability to inflict harm to the system.

Many computer devices that are seen or marketed more as embedded or managed computers than personal laptops or desktops (one example is AMD's PIC communicator) purport to dodge the issue of untrusted code entirely, while staving off viruses, malware and spyware by only permitting execution of code 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 to security certainly limits available attack vectors, it should be noted it is pointedly not a silver bullet. A computer that is not freely programmable represents a tremendous decrease in utility from what most consumers have come to expect from their computers—but even if we ignore this and focus merely on the technical qualifications of such a security system, we must stress that almost always, cryptographic signatures for binaries are checked at load time, not continually during execution. Thus exploits for vendor-provided binaries are still able to execute and harm the system. Similarly, this system fails to provide any protection against macro attacks.

As we mention in the introduction, this severely restricted execution model is absolutely not an option for the XO laptops. What's more, we want to explicitly encourage our users, the children, to engage in a scenario certain to give nightmares to any security expert: easy code sharing between computers.

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.

Unfortunately, software received through a friend or acquaintance is completely untrusted code, because there's no trust mapping between people and software: trusting a friend isn't, and cannot be, the same as trusting code coming from that friend. The friend's machine might be taken over, and may be attempting to send malicious code to all her friends, or the friend might be trying to execute a prank,

or he might have written—either out of ignorance or malice -- software that is sometimes malicious.

It is against this background that we've constructed security protections for software on the laptop. A one-sentence summary of the intent of our complete software security model is that it "tries to prevent software from doing bad things". The next chapter explains the five categories of 'bad things' that malicious software might do, and the chapter after that our protections themselves. And the chapter after it explains how each protection addresses the threat model.

H. Threat model: bad things that software can do

There are five broad categories of "bad things" that running software could do, for the purposes of our discussion. In no particular order, software can attempt to damage the machine, compromise the user's privacy, damage the user's information, do "bad things" to people other than the machine's user, and lastly, impersonate the user.

1. Damaging the machine

Software wishing to render a laptop inoperable has at least five attack vectors. It may try to ruin the machine's BIOS, preventing it from booting. It may attempt to run down the NAND chip used for primary storage, which—being a flash chip—has a limited number of write/erase cycles before ceasing to function properly and requiring replacement. Successful attacks on the BIOS or NAND cause hard damage to the machine, meaning such laptops require trained hardware intervention, including part replacement, to restore to operation. The third vector, deleting or damaging the operating system, is an annoyance that would require the machine to be re-imaged and reactivated to run.

Two other means of damaging the machine cause soft damage: they significantly reduce its utility. These attacks are performance degradation and battery drainage (with the side note that variants of the former can certainly cause the latter.)

When we say performance degradation, we are referring to the over-utilization of any system resource such as RAM, the CPU or the networking chip, in a way that makes the system too slow or unresponsive to use for other purposes. Battery drainage might be a side-effect of such a malicious performance degradation (e.g. because of bypassing normal power saving measures and over-utilization of power-hungry hardware components), or it might be accomplished through some other means. Once we can obtain complete power measurements for our hardware system, we will be aware of whether side channels exist for consuming large amounts of battery power without general performance degradation; this section will be updated to reflect that information.

2. Compromising privacy

We see two primary means of software compromising user privacy: the unauthorized sending of user-owned information such as documents and images over the network, and eavesdropping on the user via the laptops' built-in camera and microphone.

3. Damaging the user's data

A malicious program can attempt to delete or corrupt the user's documents, create large numbers of fake or garbage-filled documents to make it difficult for the user to find her legitimate ones, or attack other system services that deal with data, such as the search service. Indeed, attacking the global indexing service might well become a new venue for spam that would thus show up every time the user searched for anything on her system. Other attack vectors undoubtedly exist.

4. Doing bad things to other people

Software might be malicious in ways that do not directly or strongly affect the machine's owner or operator. Examples include performing Denial of Service attacks against the current wireless or wired network (a feat particularly easy on IPv6 networks, which our laptops will operate on by default), becoming a spam relay, or joining a floodnet or other botnet.

5. 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.

I. Protections

Here, we explain the set of protections that make up the bulk of the Bitfrost security platform, our name for the sum total of the laptop's security systems. Each protection listed below is given a concise uppercase textual label beginning with the letter P. This label is simply a convenience for easy reference, and stands for both the policy and mechanism of a given protection system.

Almost all of the protections we discuss can be disabled by the user through a graphical interface. While the laptop's protections are active, this interface cannot be manipulated by the programs on the system through any means, be it synthetic keyboard and mouse events or direct configuration file modification.

1. P_BIOS_CORE: core BIOS protection

The BIOS on an XO laptop lives in a 1MB SPI flash chip, mentioned in its section. This chip's purpose is to hold manufacturing information about the machine including its (SN, UUID) tuple, and the BIOS and firmware. Reflashing the stored BIOS is strictly controlled, in such a way that only a BIOS image cryptographically signed by OLPC can be flashed to the chip. The firmware will not perform a BIOS reflashing if the battery level is detected as low, to avoid the machine powering off while the operation is in progress.

A child may request a so-called developer key from OLPC. This key, bound to the child's laptop's (SN, UUID) tuple, allows the child to flash any BIOS she wishes, to accommodate the use case of those children who progress to be very advanced developers and wish to modify their own firmware.

2. P_BIOS_COPY: secondary BIOS protection

The inclusion of this protection is uncertain, and depends on the final size of the BIOS and firmware after all the desired functionality is included. The SPI flash offers 1MB of storage space; if the BIOS and firmware can be made to fit in less than 512KB, a second copy of the bundle will be stored in the SPI. This secondary copy would be immutable (cannot be reflashed) and used to boot the machine in case of the primary BIOS being unbootable. Various factors might lead to such a state, primarily hard power loss during flashing, such as through the removal of the battery from the machine, or simply a malfunctioning SPI chip which does not reflash correctly. This section will be updated once it becomes clear whether this protection can be included.

3. P_SF_CORE: core system file protection

The core system file protection disallows modification of the stored system image on a laptop's NAND flash, which OLPC laptops use as primary storage. While engaged, this protection keeps any process on the machine from altering in any way the system files shipped as part of the OLPC OS build.

This protection may not be disabled without a developer key, explained in P_BIOS_CORE section.

4. P_SF_RUN: running system file protection

Whereas #P_SF_CORE protects the **stored** system files, #P_SF_RUN protects the **running** system files from modification. As long as #P_SF_RUN is engaged, at every boot, the running system is loaded directly from the stored system files, which are then marked read-only.

When #P_SF_RUN is disengaged, the system file loading process at boot changes. Instead of loading the stored files directly, a COW (copy on write) image is constructed from them, and system files from *that* image are initialized as the running system. The COW image uses virtually no additional storage space on the NAND flash until the user makes modifications to her running system files, which causes the affected files to be copied before being changed. These modifications persist between boots, but only apply to the COW copies: the underlying system files remain untouched.

If #P_SF_RUN is re-engaged after being disabled, the boot-time loading of system files changes again; the system files are loaded into memory directly with no intermediate COW image, and marked read-only.

#P_SF_CORE and #P_SF_RUN do not inter-depend. If #P_SF_CORE is disengaged and the stored system files are modified, but #P_SF_RUN is engaged, after reboot no modification of the running system will be permitted, despite the fact that the underlying system files have changed from their original version in the OLPC OS build.

5. P_NET: network policy protection

Each program's network utilization can be constrained in the following ways:

- Boolean network on/off restriction
- token-bucketed 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 restriction (can bind and listen on a socket), Boolean and per-port

Reasonable default rate and transfer limits will be imposed on all non-signed programs. If necessary, different policies can apply to mesh and access point traffic. Additional restrictions might be added to this list as we complete our evaluation of network policy requirements.

6. P_NAND_RL: NAND write/erase protection

A token-bucketed throttle with burst allowance will be in effect for the JFFS2 filesystem used on the NAND flash, which will simply start delaying write/erase operations caused by a particular program after its bucket is drained. It is currently being considered that such a delay behaves as an exponential backoff, though no decision has yet been made, pending some field testing.

A kernel interface will expose the per-program bucket fill levels to userspace, allowing the implementation of further userspace policies, such as shutting down programs whose buckets remain drained for too long. These policies will be maintained and enforced by the system Security Service, a privileged userspace program.

7. P_NAND_QUOTA: NAND quota

To prevent disk exhaustion attacks, programs are given a limited scratch space in which they can store their configuration and temporary files, such as various caches. Currently, that limit is 5MB. Additionally, limits will be imposed on inodes and dirents within that scratch space, with values to be determined.

This does not include space for user documents created or manipulated by the program, which are stored through the file store. The file store is explained in a later section.

8. P_MIC_CAM: microphone and camera protection

At the first level, our built-in camera and microphone are protected by hardware: an LED is present next to each, and is lit (in hardware, without software control) when the respective component is engaged. This provides a very simple and obvious indication of the two being used. The LEDs turning on unexpectedly will immediately tip off the user to potential eavesdropping.

Secondly, the use of the camera and microphone require a special permission, requested at install-time as described in its chapter, for each program wishing to do so. This permission does not, however, 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.

This means that any benign programs which are taken over but haven't declared themselves as needing the camera or microphone cannot be used neither to turn on either, NOR to ask the user to do so!

Programs which have declared themselves as requiring those privileges (e.g. a VOIP or videoconferencing app) can instruct the system to ask the user for permission to enable the camera and microphone components, and if the request is granted, the program is granted a timed capability to manipulate the components, e.g. for 30 minutes. After that, the user will be asked for permission again.

As mentioned in the chapter on installation, 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 rather low.

9. P_CPU_RL: CPU rate limiting

Foreground programs may use all of the machine's CPU power. Background programs, however, may use no more than a fixed amount—currently we're looking to use 10%—unless given a special permission by the user.

The Sugar UI environment on the XO laptops does not support overlapping windows: only maximized application windows are supported. When we talk about foreground and background execution, we are referring to programs that are, or are not, currently displaying windows on the screen.

10. P_RTC: real time clock protection

A time offset from the RTC is maintained for each running program, and the program is allowed to change the offset arbitrarily. This fulfills the need of certain programs to change the system time they use (we already have a music program that must synchronize to within 10ms with any machines with which it co-plays a tune) while not impacting other programs on the system.

11. P_DSP_BG: background sound permission

This is a permission, requestable at install-time, which lets the program play audio while it isn't in the foreground. Its purpose is to make benign programs immune to being used to play annoying or embarrassing loud sounds if taken over.

12. P_X: X window system protection

When manually assigned to a program by the user through a graphical security interface, 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.

13. P_IDENT: identity service

The identity service is responsible for generating an ECC key pair at first boot, keeping the key pair secure, and responding to requests to initiate signed or encrypted sessions with other networked machines.

With the use of the identity service, all digital peer interactions or communication (e-mails, instant messages, and so forth) can be cryptographically signed to maintain integrity even as they're routed through potentially malicious peers on the mesh, and may also be encrypted in countries where this does not present a legal problem.

14. P_SANDBOX: program jails

A program on the XO starts in a fortified chroot, akin to a BSD jail, where its visible filesystem root is only its own constrained scratch space. 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 file store service, explained in the next section.

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.

15. P_DOCUMENT: file store service

Unlike with traditional machines, user documents on the XO laptop are not stored directly on the filesystem. Instead, they are read and stored through the file store service, which provides an object-oriented interface to user documents. Similar in very broad terms to the Microsoft WinFS design, the file store allows rich metadata association while maintaining traditional UNIX read()/write() semantics for actual file content manipulation.

Programs on the XO may not use the open() call to arbitrarily open user documents in the system, nor can they introspect the list of available documents, e.g. through listing directory contents. Instead, when a program wishes to open a user document, it asks the system to present the user with a 'file open' dialog. A copy-on-write version of the file that the user selects is also mapped into this scratch space—in effect, the file just "appears", along with a message informing the program of the file's path within the scratch space.

Unix supports the passing of file descriptors (fds) through Unix domain sockets, so an alternative implementation of #P_DOCUMENT would merely pass in the fd of the file in question to the calling program. We have elected not to pursue this approach because communication with the file store service does not take place directly over Unix domain sockets, but over the D-BUS IPC mechanism, and because dealing with raw fds can be a hassle in higher-level languages.

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 (with the rare exception of programs which create custom dialogs to e.g. offer built-in file previews; for the time being, we are not going to support this use case).

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.

16. 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—in this case, each photo.

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 viewing program, in other words, normally has no business connecting to the Internet.

As with other permissions, the user may assign the network permission to a program which requested #P_DOCUMENT_RO at install, bypassing the mutual exclusion.

17. P_DOCUMENT_RL: file store rate limiting

The file store does not permit programs to store new files or new versions of old files with a frequency higher than a certain preset, e.g. once every 30 seconds.

18. P_DOCUMENT_BACKUP: file store backup service

When in range of servers that advertise themselves as offering a backup service, the laptop will automatically perform incremental backups of user documents which it can later retrieve. Because of the desire to avoid having to ask children to generate a new digital identity if their laptop is ever lost, stolen or broken, by default the child's ECC keypair is also backed up to the server. 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. We should also mention that the private key will only be backed up to the primary backup server—usually in the school—and not any server that advertises itself as providing backup service. Furthermore, for all non-primary backup servers, only encrypted version of the incremental backups will be stored.

19. P_THEFT: anti-theft protection

The OLPC project has received very strong requests from certain countries considering joining the program to provide a powerful anti-theft service that would act as a theft deterrent against most thieves.

We provide such a service for interested countries to enable on the laptops. It works by running, as a privileged process that cannot be disabled or terminated even by the root user, an anti-theft daemon which detects Internet access, and performs a call-home request—no more than once a day—to the country's anti-theft servers. In so doing, it is able to securely use NTP to set the machine RTC to the current time, and then obtain a cryptographic lease to keep running for some amount of time, e.g. 21 days. The lease duration is controlled by each country.

A stolen laptop will have its (SN, UUID) tuple reported to the country's OLPC oversight body in charge of the anti-theft service. The laptop will be marked stolen in the country's master database.

A thief might do several things with a laptop: use it to connect to the Internet, remove it from any networks and attempt to use it as a standalone machine, or take it apart for parts.

In the former case, the anti-theft daemon would learn that the laptop is stolen as soon as it's connected to the Internet, and would perform a hard shutdown and lock the machine such that it requires activation, described previously, to function.

We do not expect the machines will be an appealing target for part resale. Save for the custom display, all valuable parts of the XO laptops are soldered onto the motherboard.

To address the case where a stolen machine is used as a personal computer but not connected to the Internet, the anti-theft daemon will shut down and lock the machine if its cryptographic lease ever expires. In other words, if the country operates with 21-day leases, a normal, non-stolen laptop will get the lease extended by 21 days each day it connects to the Internet. But if the machine does not connect to the Internet for 21 days, it will shut down and lock.

Since this might present a problem in some countries due to intermittent Internet access, the leases can either be made to last rather long (they're still an effective theft deterrent even with a 3 month duration), or they can be manually extended by connecting a USB drive to the activation server. For instance, a country may issue 3-week leases, but if a school has a satellite dish failure, the country's OLPC oversight body may mail a USB drive to the school handler, which when connected to the school server, transparently extends the lease of each referenced laptop for some period of time.

The anti-theft system cannot be bypassed as long as #P_SF_CORE is enabled (and disabling it requires a developer key). This, in effect, means that a child is free to do any modification to her machine's userspace (by disabling #P_SF_RUN without a developer key), but cannot change the running kernel without requesting the key. The key-issuing process incorporates a 14-day delay to allow for a slow theft report to percolate up through the system, and is only issued if the machine is not reported stolen at the end of that period of time.

20. P_SERVER_AUTH: transparent strong authentication to trusted server

When in wireless range of a trusted server (e.g. one provided by OLPC or the country), 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—we know about some schools, for instance, that do not wish to provide Internet access to alumni, but only current students—this authentication can unlock extra services like backup and access to a decentralized digital identity system such as OpenID.

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 by #P_SERVER_AUTH.

21. P_PASSWORD: password protection (For later implementation)

It is unclear whether this protection will make it in to generation 1 of the XO laptops. When implemented, however, it will allow the user to set a password to be used for her digital identity, booting the machine, and accessing some of her files.

J. Addressing the threat model

We look at the five categories of "bad things" software can do as listed in the corresponding chapter, and explain how protections listed in their chapter help. The following sections are given in the same order as software threat model entries in their chapter.

1. Damaging the machine

#P_BIOS_CORE ensures the BIOS can only be updated by BIOS images coming from trusted sources. A child with a developer key may flash whichever BIOS she pleases, though if we are able to implement #P_BIOS_COPY, the machine will remain operational even if the child flashes a broken or garbage BIOS. Programs looking to damage the OS cannot do so because of #P_SANDBOX and #P_SF_RUN. Should a user with #P_SF_RUN disabled be tricked into damaging her OS or do so accidentally, #P_SF_CORE enables her to restore her OS to its initial (activated) state at boot time.

Programs trying to trash the NAND by exhausting write/erase cycles are controlled through #P_NAND_RL, and disk exhaustion attacks in the scratch space are curbed by #P_NAND_QUOTA. Disk exhaustion attacks with user documents are made much more difficult by #P_DOCUMENT_RL.

CPU-hogging programs are reined in with #P_CPU_RL. Network-hogging programs are controlled by policy via #P_NET.

2. Compromising privacy

Arbitrary reading and/or sending of the user's documents over the network is curbed by #P_DOCUMENT, while tagging documents with the program that created them addresses the scenario in which a malicious program attempts to spam the search service. Search results from a single program can simply be hidden (permanently), or removed from the index completely.

#P_DOCUMENT_RO additionally protects the user from wide-scale privacy breaches by software that purports to be a "viewer" of some broad class of documents.

#P_MIC_CAM makes eavesdropping on the user difficult, and #P_X makes it very hard to steal passwords or other sensitive information, or monitor text entry from other running programs.

3. Damaging the user's data

File store does not permit programs to overwrite objects such as e-mail and text which aren't opaque binary blobs. Instead, only a new version is stored, and the file store exposes a list of the full version history. This affords a large class of documents protection against deletion or corruption at the hands of a malicious program—which, of course, had to obtain the user's permission to look at the file in question in the first place, as explained in #P_DOCUMENT.

For binary blobs—videos, music, images—a malicious program in which the user specifically opens a certain file does have the ability to corrupt or delete the file. However, we cannot protect the user from herself. We point out that such deletion is constrained to *only* those files which the user explicitly opened. Furthermore, #P_DOCUMENT_BACKUP allows a final way out even in such situations, assuming the machine came across a backup server (OLPC school servers advertise themselves as such).

4. Doing bad things to other people

XO laptops will be quite unattractive as spam relays or floodnet clients due to network rate and transfer limits imposed on all non-signed programs by #P_NET. Despite the appeal of the XO deployment scale for spamming or flooding, we expect that a restriction to generally low-volume network usage for untrusted software—coupled with the great difficulty in writing worms or self-propagating software for XO machines—will drastically reduce this concern.

5. Impersonating the user

The design of the identity service, #P_IDENT, does not allow programs to ever come in direct contact with the user's cryptographic key pair, nor to inject information into currently-open sessions which are using the identity service for signing or encryption.

6. Miscellaneous

In addition to the protections listed above which each address some part of the threat model, permissions #P_RTC and #P_THEFT combine to offer an anti-theft system that requires non-trivial sophistication (ability to tamper with on-board hardware) to defeat, and #P_DSP_BG provides protection against certain types of annoying malware, such as the infamous 1989 Yankee Doodle virus.

7. Missing from this list

At least two problems, commonly associated with laptops and child computer users respectively, are not discussed by our threat model or protection systems: hard drive encryption and objectionable content filtering / parental controls.

a. Filesystem encryption

While the XO laptops have no hard drive to speak of, the data encryption question applies just as well to our flash primary storage. The answer consists of two parts: firstly, filesystem encryption is too slow given our hardware. The XO laptops can encrypt about 2-4 MB/s with the AES-128 algorithm in CBC mode, using 100% of the available CPU power. This is about ten times less than the throughput of the NAND flash chip. Moving to a faster algorithm such as RC4 increases encryption throughput to about 15 MB/s with large blocks at 100% CPU utilization, and is hence still too slow for general use, and provides questionable security. Secondly, because of the age of our users, we have explicitly designed the Bitfrost platform not to rely on the user setting passwords to control access to her computer. But without passwords, user data encryption would have to be keyed based on unique identifiers of the laptop itself, which lends no protection to the user's documents in case the laptop is stolen.

Once the Bitfrost platform supports the #P_PASSWORD protection, which might not be until the second generation of the XO laptops, we will provide support for the user to individually encrypt files if she enabled the protection and set a password for herself.

b. Objectionable content filtering

The Bitfrost platform governs system security on the XO laptops. Given that "objectionable content" lacks any kind of technical definition, and is instead a purely social construct, filtering such content lies wholly outside of the scope of the security platform and this document.

K. Laptop disposal and transfer security

The target lifetime of an XO laptop is five years. After this time elapses, the laptop's owner might wish to dispose of the laptop. Similarly, for logistical reasons, a laptop may change hands, going from one owner to another.

A laptop re-initialization program will be provided which securely erases the user's digital identity and all user documents from a laptop. When running in "disposal" mode, that program could also be made to permanently disable the laptop, but it is unclear whether such functionality is actually necessary, so there are no current plans for providing it.

L. Closing words

In Norse mythology, Bifröst is the bridge which keeps mortals, inhabitants of the realm of Midgard, from venturing into Asgard, the realm of the gods. In effect, Bifröst is a powerful security system designed to keep out unwanted intruders.

This is not why the OLPC security platform's name is a play on the name of the mythical bridge, however. What's particularly interesting about Bifröst is a story that 12th century Icelandic historian and poet Snorri Sturluson tells in the first part of his poetics manual called the Prose Edda. Here is the relevant excerpt from the 1916 translation by Arthur Gilchrist Brodeur:

Then said Gangleri

> "What is the way to heaven from earth?"

Then Hárr answered, and laughed aloud

> "Now, that is not wisely asked; has it not been told thee, that the gods made a bridge from earth, to heaven, called Bifröst? Thou must have seen it; it may be that ye call it rainbow.' 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."

Then said Gangleri

> "To my thinking the gods did not build the bridge honestly, seeing that it could be broken, and they able to make it as they would."

Then Hárr replied

> "The gods are not deserving of reproof because of this work of skill: a good bridge is Bifröst, but nothing in this world is of such nature that it may be relied on when the sons of Múspell go a-harrying."

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 we've imbued the OLPC security system with cunning and more magic art than other similar works of craftmanship—but not for a second do we believe we've designed something that cannot be broken when talented, determined and resourceful attackers go forth harrying. Indeed, 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.