SAFE-OS: a Secure and Usable Desktop Operating System

François Lesueur*, Ala Rezmerita*, Thomas Herault*, Sylvain Peyronnet* and Sébastien Tixeuil†

*LRI, University Paris-Sud
Orsay, France
Email: firstname.lastname@lri.fr
†LIP6, UPMC Paris Universitas
Paris, France
Email: firstname.lastname@lip6.fr

***Abstract*—Containment of application execution is a key security feature of operating systems. Without strong containment, an attacker who compromises one process may take control of the whole machine. Virtualization technology has been widely used in server systems to strongly isolate various applications or services in different virtual machines; its usage in desktop systems which are much more interactive (interactions with the user and between applications) is a challenging task. In this paper we describe SAFE-OS, a desktop operating system using virtualization technology. SAFE-OS provides a high level of isolation between processes while maintaining a standard user interface that abstracts the underlying complexity.**

I. Introduction

Containment of application execution is a key security feature of operating systems. However, to remain usable, a system cannot enforce full containment of the data and applications. Indeed, it is mandatory that a user can access the same data with different applications.

At the same time, vulnerabilities in complex and large applications are for the most part unavoidable. An attacker who has the opportunity to attack an application can take the control of this application and obtain the same privileges as this application has on the attacked computer.

An approach that has been taken in many server-oriented systems is to avoid the propagation of the attack by isolating the different services on multiple different physical machines. Using virtualization, this isolation can also be obtained by running multiple virtual machines on a single physical machine. As long as the virtualization software effectively isolates the different virtual machines, corrupted applications that run under a given virtual machine cannot interact with the other applications running on other VMs.

Obviously, the security of desktop systems would also benefit from the strong containment properties that are provided by virtualization. However, desktop systems must exhibit a unified global interface from which the user can interact with all her applications that in turn, need to share files. Leveraging the security level offered by the strong containment of virtualization in such an interactive system is thus a challenging problem.

In SAFE-OS, we tackle this problem and propose a desktop operating system providing virtualization-based isolation. The overall impression of the user remains essentially similar to that of a standard OS, even if the system distributedly runs on several virtual machines. Moreover, SAFE-OS enforces fine-grained control over the tasks running on the different virtual machines. SAFE-OS provides a standard desktop interface and allows the necessary interactions between the virtual machines to maintain an enjoyable user experience while providing a high security environment.

We consider the following threat model. Attackers are external to the system and can thus only perform network attacks (they have no physical access to the machine). On the other side, we consider that the user is well-intentioned but is not a computer specialist.

The remaining of the paper is organized as follows. In Section II, we present related work. Section III presents the overall architecture of SAFE-OS and Section IV describes its implementation. In Section V, we discuss how files can be shared by applications. In Section VI, we evaluate the security and performance of SAFE-OS. Section VII provides some concluding remarks.

II. Related Work

In this section, we present work related to containment of applications. We first present kernel-based protections, then we describe virtualization-based protections.

A. Kernel-Based Containment

The classic solution to isolate different activities one from another is the use of access rights such as HRU[1]. Each user is then only allowed to access a subset of files in the system. However, the corruption of an application allows the attacker to access every file of the user or to change permissions on these files.

Mandatory Access Control (MAC) systems like SELinux[2], TOMOYO Linux[3] or AppArmor[4] allow to limit the damage a compromised application can cause. This security feature allows the administrator to set fine grained access control policy to the system, thus limiting the privileges an attacker would

978-1-4244-8642-7/10/$26.00 ©2010 IEEE

gain by diverting an application. However, MAC solutions have a high administrative cost since the policies are large and complex and therefore difficult to maintain. Moreover, they depend on the kernel security, kernel which is a very complex piece of software and thus prone to some vulnerabilities[1].

B. Virtualization-Based Containment

Virtualization technologies, such as Xen[5], allow to contain different domains in different Virtual Machines (VMs). Since the virtualization code is much simpler than a modern kernel code, its security level can be much higher[2] and virtualization has thus been used to securely contain different applications.

SVFS[6] and Storage Capsules[7] aim to protect sensitive data. In SVFS, multiple virtual machines run on a single physical machine, one of which mediates file access from other VMs. SVFS only protects the integrity of some sensitive system files but does not protect the confidentiality nor integrity of user data. A corrupted application can then access personal data and alter them or send them to the attacker. In Storage Capsules, when the user wants to view or edit files, the system is switched into secure mode, a checkpoint of system state is taken and the network device output is disabled. When the user has finished editing the sensitive file, Capsule re-encrypts it with an isolated module, restores the whole system to its original state and re-enables the network device output. Thus, even a compromised machine is not able to steal sensitive files contents. The major drawback of Capsule system is that it is designed to protect files that are edited locally and cannot provide security in an interactive context, such as an online banking session. Moreover, critical operations cannot be conducted concurrently to standard operations.

NetTop[8] uses VMware to run multiple isolated VMs, each one having its own security classification level and being unable to access data with different security levels. However, NetTop provides complete isolation, which is not suitable in our context: an ordinary user may need to process files obtained or created by a given application with another application running in a different VM.

Bitfrost[9], used in the OLPC initiative, and Qubes OS[3] are the projects closest to ours. Each application runs in its own VM and is protected from corruptions originating in other VMs. All applications, although running contained, are displayed in a unified interface. In Bitfrost, all personal data are stored in a central directory, isolated from other VMs. However, applicative VMs have no direct access to the central directory and must use some dedicated primitives to open files: this constrains to rewrite every file open procedure in all applications specifically for Bitfrost. Moreover, Bitfrost provides some generic capabilities for VMs (Internet, webcam, ...) but does not allow fine-grained control on the VMs. In Qubes OS, each VM runs independently and data can be stored in every VM. Although the current release of Qubes OS is technically well advanced, the authors do not define the roles of the different VMs; instead, each VM can indifferently run any application. Qubes OS thus do not propose fine-grained control of each VM based on its intended use. Qubes OS also does not tackle the security implications of migrating files among VMs.

In this paper, we describe SAFE-OS in which the different applications are contained in different VMs. We tackle the usability problem and the virtualization layer is made transparent by displaying the applications in a unified interface. It means that we propose a user experience as close as possible to the one with a usual OS. Moreover, SAFE-OS uses unmodified applications, proposes fine-grained control on the interactions of these applications and allows to run both critical and standard applications concurrently.

III. Architecture

The security of SAFE-OS is twofold. First, using virtualization, each application runs in its own contained environment and the corruption of an application is limited to this peculiar environment. Second, since each contained environment is dedicated to specific tasks, its security policy is tightened to make its corruption even harder. In this section, we first present the containment obtained through virtualization. We then define the different parts of our solution, namely the *Base*, the *Main Env* and the *Appliances*.

A. Containment through Virtualization

Virtualization technologies allow to run different Virtual Machines on a single physical machine. Each VM runs on its own virtualized hardware (main memory, hard disk, network card, ...) and is thus isolated from the other VMs. For instance, a VM cannot access the main memory of another VM nor preempt the CPU cycles attributed to it.

This isolation ensures a strong security property: the containment of attacks. Although the VMs are running on the same hardware, the corruption of a single one, using a vulnerability of a running application, does not allow the attacker to compromise the other VMs. For instance, if a web browser is running on a VM and is corrupted by a malicious website exploiting a 0-day vulnerability (or even a kernel vulnerability), the mail client running on another VM is unaffected and the attacker cannot obtain confidential mails nor address books.

However, applications must still interact with the outer world and with the user: these two interfaces are thus shared by all the VMs. From a security point of view, these interfaces must maintain the containment between VMs and, from the user point of view, they must give him the ability to use this set of VMs as a single machine.

The architecture of SAFE-OS is illustrated in Figure 1 and is composed of three parts. First, the VM *Base* ensures the communications between the different VMs and the outer world and enforces communications policies. Then, the VM *Main Env* is the only environment the user interacts with. This environment displays all the applications, which are run on different VMs, on a single interface, thus providing the user

[1] http://secunia.com/product/2719/?task=statistics
[2] http://secunia.com/product/15863/?task=statistics
[3] http://qubes-os.org/files/doc/arch-spec-0.3.pdf

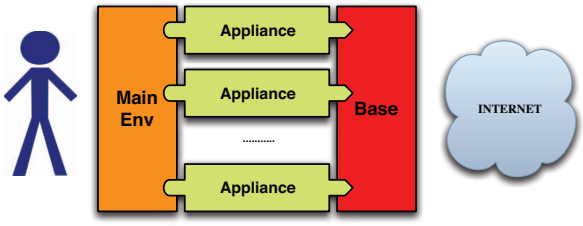

Fig. 1. Global architecture of SAFE-OS

a similar experience to a standard OS. Finally, a set of VMs *Appliances* run the different applications on contained environments. The user interacts with these *Appliances* through the *Main Env* and these *Appliances* communicate with the outer world through the *Base*.

B. Roles of the different VMs

1) Base: VM/World Communication Enforcer: In SAFE-OS, the *Base* is used to implement the communication policy between the appliances and the outer world. It mainly consists of an applicative firewall which provides hooks for appliance-linked security modules.

The *Base* is the only VM having access to the Internet and is the only VM accessible from every other VMs. Every communication going out from another VM, either destined to a local VM or to the Internet, go thus through the *Base* which can filter them.

2) Main Env: VM/User Communication Center: The *Main Env* is the interface between the appliances and the user. This environment is the only one the user has direct access to and allows to control and display all the appliances. This interface abstracts the multiple appliances on a single screen and the user has no knowledge of the underlying virtualization infrastructure.

C. Appliances: User Service Providers

Each appliance is responsible for providing a specific service to the user. An appliance is an autonomous VM and is thus composed of a minimalistic OS running the application provided by the appliance.

Each appliance is plugged into the *Base* (security module) and into the *Main Env* (shortcuts). Appliances are then controlled from and displayed to the *Main Env*; they may communicate over the Internet through the *Base*, if allowed by the security policy. The execution of the application is contained in the appliance VM (only the display is exported to the *Main Env*) and thus cannot corrupt nor be corrupted by other appliances.

We distinguish two main types of appliances: *Critical Appliances* which host secured services, are persistent and may contain personal data of the user and *Sandboxes* which host untrusted services, are meant to be easily reinstalled and should not be used to store important personal data (these data should only be there for transient periods).

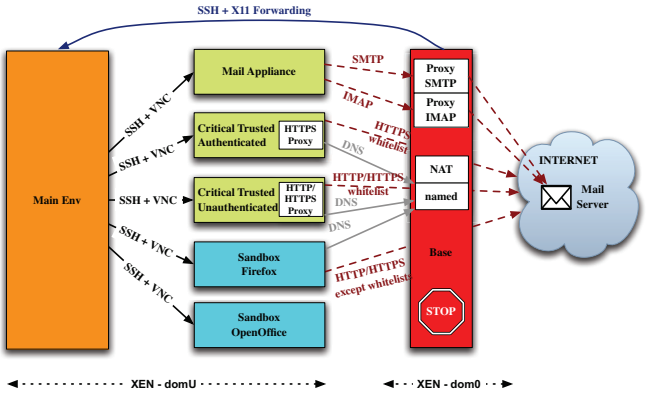

Fig. 2. Communications among the VMs

IV. Implementation

The containment has been implemented in SAFE-OS using Xen [5], a virtual machine monitor. The XEN hypervisor is a minimalistic OS running directly on the physical machine. On top of this hypervisor, several VMs can be hosted on a single physical machine. As other virtualization tools, XEN uses X86-virtualization (Intel VT and AMD-V) to achieve nearly native performance. Each VM runs Debian GNU/Linux.

The first booted virtual machine is called the *dom0*. The *dom0* automatically obtains some privileges, can directly access the hardware and can start other virtual machines: the Base Environment is naturally instantiated into this *dom0*. The other virtual machines, called *domU* (Unprivileged), run with no privileges on top of virtualized hardware. The *Main Env* and all appliances run as *domU*. A global view of the communication policy is shown in Figure 2.

A. Instantiation of the VMs

1) Base: XEN *dom0* runs the *Base* Virtual Machine. It provides network services to other VMs (NAT, DNS) and executes the main X server, since it is the only VM having access to the physical hardware (keyboard, mouse and graphic card). This X display is then delegated to the *Main Env*.

Since all communications between and from VMs are routed through the *Base*, it runs an *iptables* firewall to implement communication policies and to NAT outgoing connections. The *Base* itself is nearly transparent and mostly forwards packets from the other VMs. It only accepts incoming connections to specific proxy services that implement the security modules of the appliances. This transparency greatly limits the attack possibilities towards the *dom0* system.

Security modules of appliances run as unprivileged processes on this *dom0*. The firewall only allows the associated appliance to communicate with them on specified ports.

2) Main Env: The *Main Env* is responsible for the user interface and holds a user-friendly GNOME graphical desktop and window manager. This environment has no Internet access and can only communicate with appliances to control and display them.

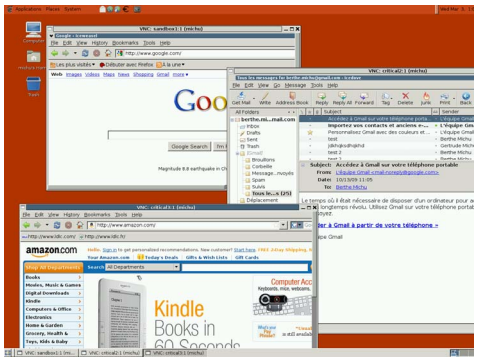

Fig. 3. The SAFE-OS interface is similar to a standard OS although the different applications are executed on different VMs.

The *Main Env* is illustrated on Figure 3. Each appliance is displayed in its own window, as if it was executed locally. The graphical interface totally abstracts the underlying complexity and virtualization.

3) Critical Appliances: We present here the three appliances hosting critical services. These appliances allow to read and write emails, to navigate on trusted and authenticated websites and to navigate on trusted but not authenticated websites.

a) Mail: This VM runs Mozilla Thunderbird for email management, uses the IMAP protocol to fetch mails and SMTP to send mails. This appliance ensures that even if it is corrupted, through a malicious email exploiting a vulnerability of Thunderbird for instance, the attack is contained in this appliance and the personal emails of the user cannot be disclosed to the attacker. The containment is ensured by the virtualization and a security module running on the *Base*, which is considered uncorrupted, manages the security of contained data. The security module runs under an unprivileged user on the *Base* VM. It mainly consists of IMAP and SMTP proxy services. The Mail VM has absolutely no access to the Internet (not even DNS, which would provide a covert channel) and can only connect to its proxy services on the *Base*.

The IMAP proxy enforces that the IMAP connection is established to the right server with proper username. These constraints, ensured by the *Base*, prevent information leakage through the IMAP service if the Mail appliance is corrupted. Indeed, the Mail VM can only communicate with the remote mailbox of the user. This IMAP proxy uses `Perdition`[4] and `stunnel`[5]. `Perdition` listens for connections from the Mail appliance and constrains the username sent to the remote server to be the proper username; `stunnel`, an SSL tunnel, connects to the right server using SSL (IMAPS), asserts that the remote certificate is valid (protection from Man-in-the-Middle attacks) and tunnels the IMAP connection to this remote host. The only remaining path for data leakage is then through the SMTP service.

The SMTP proxy enforces that the SMTP connection is established with the right server and that the mail was indeed sent by the user (thus, a corrupted Mail appliance cannot send mails in the background). This SMTP proxy uses `ProxSMTP`[6] with a custom filter and `stunnel`. `ProxSMTP` listens for SMTP connections from the mail appliance. When such a connection is established, it blocks it and checks the recipients of this mail. If the recipients were previously whitelisted, the sending continues; else, a confirmation popup is presented to the user, containing the list of the recipients, asking the user to check them and proposing to whitelist some of them and to proceed. Since the SMTP proxy resides in the *dom0*, it is isolated from a corrupted Mail appliance; moreover, the popup is handled by the *dom0* which runs the X11 server showed to the user and a corrupted Mail appliance, which runs on another X11 server on another VM, cannot interact with this window. The SMTP connection is then established through `stunnel`, as previously.

b) Trusted Authenticated Websites: This VM runs Mozilla Firefox to navigate on websites that are both trusted and authenticated. This appliance can only connect to whitelisted websites (trusted) using the `https` protocol (authenticated). In this appliance, the user can only browse *safe* content, as long as the hypothesis that whitelisted websites are honest holds, and thus this appliance cannot be corrupted when browsing the web. The confidentiality and integrity of data stored in this appliance is thus guaranteed.

The list of authorized websites is enforced by `Tinyproxy`[7], a filtering proxy running in the appliance itself. Indeed, since the appliance remains clean as long as it is prevented from browsing unsafe websites, this control can be handled in the appliance instead of in the *Base* VM, to keep the *Base* VM minimalistic. To avoid any Man-in-the-Middle attack, no security exceptions are allowed (invalid certificates are rejected).

c) Trusted but Unauthenticated Websites: This VM runs Mozilla Firefox and is quite similar to the aforementioned one. The main difference is that it is allowed to connect to both ports 80 (`http`) and 443 (`https`). This type of communication is required, for instance, on e-selling websites which propose to shop using only the `http` protocol and then use `https` to finalize the transaction and pay the order. The list of allowed websites is enforced using `Tinyproxy`.

Compared to the preceding appliance, this appliance has no strong guarantee to be communicating with the intended remote host. Indeed, an attacker may set up a Man-in-the-Middle attack on a `http` connection. On the one hand, allowing unsecured `http` traffic, this appliance must be isolated from the preceding one; on the other hand, in the absence of Man-in-the-Middle attacks (which are not easy to setup on the Internet), this appliance provides the same security properties as the preceding one. This appliance provides thus a much higher security level than sandboxes.

[4]http://www.vergenet.net/linux/perdition/
[5]http://www.stunnel.org/
[6]http://memberwebs.com/stef/software/proxsmtp/
[7]https://www.banu.com/tinyproxy/

4) Sandboxes: We present here the two sandboxes, appliances which allow to execute risky tasks without compromising the security of other VMs. The first one allows to navigate on untrusted websites and the second one is used to work on documents.

a) Untrusted Websites: This VM runs Mozilla Firefox and is used to browse ordinary websites. It has access to the whole Internet on both `http` and `https` protocol. No filtering is applied on the requested URLs and the *Base* also provides a DNS service to this appliance.

b) Office Applications: This VM runs OpenOffice and allows the user to edit documents in a secured environment. This VM has no Internet access allowed, not even DNS, since it is not needed to edit documents. The data transfer tool presented in Section V is used to transfer files from and to this appliance; files can then be opened safely in this sandbox without loosing confidentiality, even if some of them contain viruses or other malicious codes.

B. Global Functioning

1) Communications between VMs: The different VMs communicate using SSH and VNC. An SSH tunnel is created at login time between the X-server running on the *Base* and the *Main Env* running the desktop and window manager. SSH authentication is established using cryptographic keys and the private key of the user is stored on the *Base* and loaded in an SSH agent.

SSH is then used to control the appliances from the *Main Env* by launching necessary applications. To maximize the containment between appliances, each appliance runs its own local X-server and interacts only with this local X server: VNC is then used to interact with the appliances from the *Main Env*.

The *Main Env* uses a VNC viewer to interact with the appliances. As a consequence, all inputs on one VNC viewer are contained to this single window, and as long as the VNC viewer is not corrupted (it only displays pixels and sends events), a process that takes control of a VNC server in an appliance cannot interfere with the interactions with the other VNC viewers connected to other appliances.

2) Automatic Updates: During the boot process, the *Base* VM checks for operating system updates and possibly updates the different VMs.

To maintain restrictions on Internet access for the VMs and to prevent the possible information leakage, the automatic update is performed at boot time. For each VM, a file system skeleton is present on the physical machine and is updated during the boot; it is then synchronized with the real VM file system, excluding user personal data.

This method ensures that even if some malicious software has been installed on the file system of a VM, using a vulnerability of an appliance, it is removed during the update. SAFE-OS is thus kept clean from malicious softwares.

V. Data Transfer

The containment of the different applications allows to limit the impact of a successful attack. However, the containment of the applications also implies the containment of their manipulated data, which would be a burden to the usability of SAFE-OS. In this section, we describe this problem and propose a workaround.

A. Limitations Due to the Containment

In SAFE-OS, each application runs in its own VM (appliance) and thus, on its own OS and (virtual) hard disk. This virtualization implements the necessary containment and ensures that the corruption of an application has no impact on the other applications (hosted by other appliances) and their data.

However, the user often needs to process the files obtained or created by a given application with another application. If this other application is hosted by another appliance, it has no access to these files. For instance, the user may need to work on a spreadsheet received by email. This spreadsheet is received as an attachment and, as such, can be stored on the mail appliance. However, the Office suite is hosted by another appliance and the user thus needs to migrate this file from one appliance to another.

The migration of files between appliances is clearly antagonistic to the containment. These migrations must thus be carefully controlled. We believe that automatic control is not pertinent: since the computer has no knowledge on the semantic of the files, it cannot decide accurately for their migration and the policy would be too tightened. The user must thus be put at the center of the decision process and the computer must provide her enough information to help her taking the right decision.

B. A Security Compliant Data Transfer Tool

In order to circumvent the limitations introduced by the containment, we propose a security compliant data transfer tool. This graphical tool allows the user to migrate files from one appliance to another and helps her taking the right decision aware of the possible consequences.

1) Characterization of Appliances: We characterize the appliances through their network connectivity and the type of data they contain.

Each appliance has either Internet access (full or restricted) or no connectivity at all. We consider that an appliance has Internet connectivity if it can access at least some Internet services; we consider that an appliance has no connectivity if it cannot connect to any Internet service (firewalled from *dom0*).

Then, each appliance is tagged as containing critical data or not. Critical appliances are tagged as containing critical data and sandboxes are initially tagged as not containing critical data. Then, when critical data are transferred to a sandbox, this sandbox becomes tagged as containing critical data until it is reset.

2) Warning Types: We consider two warning types when migrating a file f from an appliance A_1 to an appliance A_2: the confidentiality of files in A_1, including f, and the integrity of files in A_2. The migration of f has no impact on the integrity

A_2 \ A_1	Critical	Not Critical
Critical/Internet	$C(A_1)$, $I(A_2)$	$I(A_2)$
Critical/No Internet	$I(A_2)$	$I(A_2)$
Not Critical/Internet	$C(A_1)$	$\emptyset$
Not Critical/No Internet	$\emptyset$	$\emptyset$

TABLE I
FILE MIGRATION POLICY. FOR A FILE TRANSFERRED FROM $\boldsymbol{A_1}$ TO $\boldsymbol{A_2}$, THE IMPACTS CAN BE THE CONFIDENTIALITY OF $\boldsymbol{A_1}$, DENOTED $\boldsymbol{C(A_1)}$, OR THE INTEGRITY OF $\boldsymbol{A_2}$, DENOTED $\boldsymbol{I(A_2)}$.

of files in the source appliance A_1 (including f) nor on the confidentiality of files in the target appliance A_2. In fact, there is no information flow *to* A_1 (no possible data corruption) nor *from* A_2 (no possible data leakage).

When migrating the file f, it is automatically exposed to disclosure if the target appliance A_2 is corrupted and if A_2 has some Internet access. If the file f comes from an appliance A_1 tagged as containing critical data and if A_2 has some Internet access (restricted or full), then the user must be warned that the secrecy of the file f is not ensured anymore after a migration and A_2 becomes tagged as containing critical data.

If the source appliance A_1 is corrupted but has a restricted access to the Internet, the attacker may not be able to obtain files stored in this appliance. This may be the case, for instance, in the Mail appliance: the attacker may corrupt this machine through a virus emailed to the victim but the extraction of files contained in this appliance requires the explicit cooperation of the victim. In that case, a file f may contain information of other files which should be locked in the A_1 appliance. If the A_1 appliance is tagged as containing critical data and if the A_2 appliance has some Internet access (restricted or full), the user must be warned that this migration may expose other files of A_1. Moreover, if A_1 was tagged as critical, A_2 becomes tagged as critical too.

Finally, if the source appliance A_1 is corrupted, the file f may contain some malicious code. This file can thus alter the integrity of other files in the target appliance A_2. If A_2 is tagged as containing critical data, the user must be warned that, if A_1 is corrupted, the migration of f may corrupt A_2.

The possibly impacted resources are summed up in Table I. In a standard usage, we expect that most of the migrations will occur from a critical appliance (web or mail) to an Office sandbox without Internet access, in order to work on downloaded files. This case, for instance, does not trigger any warning as long as the sandbox is regularly reset.

VI. EVALUATION

SAFE-OS has been implemented and is freely available on http://safe-os.lri.fr. Each VM runs Debian GNU/Linux Lenny (stable 5.05) which uses kernel version 2.6.26 and XEN version 3.2. In this section, we evaluate the security of SAFE-OS and we benchmark its performance.

A. Security Evaluation

In this subsection, we evaluate the security of SAFE-OS. First, we analyze the interactions among the VMs and we show that these interactions maintain the containment needed for the security. Then, we describe how SAFE-OS can restore itself after an intrusion, as long as the *dom0* remains clean. Finally, we explain the external evaluation conducted on SAFE-OS.

1) Interactions between VMs: Xen is responsible for the isolation of the different VMs on a single physical machine. Xen has been designed especially to isolate different VMs with security in mind. Moreover, since the Xen hypervisor can be seen as a very minimalistic OS (less than 150,000 lines of code), its careful design allows a high level of security.

VNC is used to display and interact with the appliances. Contrary to a remote X server or to a SSH X11 forwarding, in which case the X events are interpreted locally, the VNC protocol only transfers the image of the display and the keyboard/mouse events: the VNC client only displays received images and do not interpret anything in them. VNC allows thus a high level of containment between the appliances and the *Main Env*.

SSH is used to control the appliances from the *Main Env*. SSH is highly secured and is thus reliable to protect the *Main Env* from corruptions originating at the appliances.

Security modules are proxy services linked to a specific appliance and running on the *Base*. If an appliance becomes corrupted and if a proxy service contains a vulnerability, an attacker may gain access to the *Base* (Xen *dom0*). Security modules must thus be carefully designed and be kept simple to allow their safe coding.

2) Resilience: As long as the *Base* is not corrupted, SAFE-OS automatically restores corrupted VMs to a safe state. If the *Main Env* or an appliance becomes corrupted, the attacker can alter this VM in order to maintain the corruption across reboots, as in usual OS. However, as soon as the vulnerability is discovered and corrected upstream, the automatic update process updates the template contained in *dom0* and restores a clean VM.

The *dom0* of SAFE-OS allows thus to maintain a usable system, even after a successful attack, as long as the *dom0* has not been corrupted.

3) External Security Evaluation: SAFE-OS is developed in the context of a French ANR challenge. Three teams develop concurrent solutions to provide a secure Internet experience to ordinary users and attack the two other solutions. Our opponents in this challenge are OS security experts and proposed attacks, which were mostly theoretic, have been fixed. The evaluation of SAFE-OS by external experts has thus led to improvements.

SAFE-OS has thus been carefully designed and its security evaluation allowed to confirm the design choices and to improve some technical parts.

B. Benchmarking

SAFE-OS provides an interface similar to an ordinary OS. To evaluate its usability, we benchmarked its performance to

	Reference	SAFE-OS
Boot time (seconds)	42.6	107.3
Cold Firefox launch (seconds)	5.9	11.4
Warm Firefox launch (seconds)	1.5	5.2
CSS (ms)	53.3	54
SunSpider (ms)	3821	3859

TABLE II
SAFE-OS PERFORMANCE.

assert that the system is usable. We compare SAFE-OS to a stock Debian system, called *Reference*, under the same version (Debian GNU/Linux Lenny, stable 5.04). All the tests are run in virtual machines using Parallels, which means that the whole SAFE-OS runs itself as a VM.

We used the following tests, which results are synthesized in Table II. The *boot time* is the time from power-on to an available desktop (lower is better); the *Firefox launch time* is the time to launch Firefox (lower is better), for both cold start (first start after boot) and warm start (subsequent starts); the *CSS* is a CSS benchmark run in Firefox[8] (lower is better); the *SunSpider* is a JavaScript benchmark from the Webkit project[9] (lower is better).

The *boot time* and *Firefox launch time* provide clues on the overhead introduced by the presence of different VMs. During the boot of the physical machine, the *Main Env* as well as the appliances must be booted, which takes more time than the boot of a single machine. The SAFE-OS boot process could be improved by starting the appliances VMs after the boot of the *Base* and the *Main Env*. The Firefox launch time, although higher in SAFE-OS than in the *Reference* system, remains reasonably low.

The CSS and SunSpider synthetic benchmarks provide a performance evaluation of Firefox running either in a critical appliance or in a sandbox (results are similar in both cases). These benchmarks show that, after the initial launch time overhead, the applications attain the same speed in SAFE-OS than in the *Reference* system. The virtualization layers have thus a negligible impact on the speed and reactivity of the applications.

We can conclude that even if SAFE-OS introduces some overhead, its global performance remains acceptable and running applications are not noticeably slowed down, compared to the *Reference* System.

VII. CONCLUSION

We presented SAFE-OS, a secure desktop operating system that uses virtualization to ensure the confidentiality, integrity and availability of data and services for a end user. SAFE-OS is based on a modular design featuring two fundamental environments and a variable number of appliance environments. The Base Environment is responsible for enforcing the communication policy and basic features while the Main Environment is the root of all interactions with the user. Appliance Environments define additional communication rules at the *Base* level, insert shortcuts and display capabilities in the Main Environment and isolate the appliance from any interactions with the rest of the system.

In this paper, we demonstrated that SAFE-OS architecture provides both a unified user interface (although the environments are distributed in various virtual machines) and the capacity to easily exchange data between applications. After the operating system has boot, application benchmarks feature only a very slight overhead, yet provide extensive additional security. Further reducing the overhead at boot time and enhancing provided security feedback to users is an immediate path for future research.

[8]http://nontroppo.org/timer/csstest.html

[9]http://www2.webkit.org/perf/sunspider-0.9/sunspider.html

REFERENCES

[1] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communication of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[2] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the linux operating system," in *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 29–42.

[3] T. Harada, T. HORIE, and K. TANAKA, "Task Oriented Management Obviates Your Onus on Linux," 2004.

[4] M. Bauer, "An introduction to novell apparmor. linux journal , no. 148, pp. 36, 38, 40-41. august," 2006.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[6] X. Zhao, K. Borders, and A. Prakash, "Towards protecting sensitive files in a compromised system," in *SISW '05: Proceedings of the Third IEEE International Security in Storage Workshop*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 21–28.

[7] E. V. Weele, B. Lau, and A. Prakash, "Protecting confidential data on personal computers with storage capsules," in *Proceedings of the 18th USENIX Security Symposium*, 2009.

[8] R. Meushaw and D. Simard, "Nettop: Commercial technology in high assurance applications. tech trend notes: Preview of tomorrow's information technologies, 9(4), september 2000."

[9] I. Krstic and S. L. Garfinkel, "Bitfrost: the one laptop per child security model," in *Proceedings of the 3rd Symposium on Usable Privacy and Security, SOUPS 2007, Pittsburgh, Pennsylvania, USA, July 18-20, 2007*, ser. ACM International Conference Proceeding Series, L. F. Cranor, Ed., vol. 229. ACM, 2007, pp. 132–142. [Online]. Available: http://doi.acm.org/10.1145/1280680.1280697