Delivering Interactive Real-Time Multi-Media Content with a Secure Operating System

Lenny Nilsson

March 15, 2011
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Thomas Hellström
Supervisor at Dohi Sweden: Mikael Jonsén, Emanuel Dohi
Examiner: Frank Drewes

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Delivering Interactive Real-Time Multi-Media Content with a Secure Operating System

Abstract

The company Dohi Sweden is developing a user interface product. This interface will deliver interactive content on a multi-touch screen surface. The system will provide a user-friendly way to populate touch-screen surfaces with interactive real-time multi-media contents. The interactive surface is populated by a client computer, connected to the internet. Third party applications will render interactive multi-media content on the touch-screen.

This thesis report will provide an operating system (OS), for the client computer, that display interactive multi-media content. Known technologies are presented in the report. These technologies are considered as components for the construction of the OS. The client computer can be a target for different malicious software and penetration attempts. I will address the security risks and different aspects of modern computer security. A proof-of-concept OS will be a subject for security evaluation.
Contents

1 Introduction ...................................... 1

2 Problem Description ............................ 3
  2.1 Problem Statement ............................ 3
  2.2 Purposes ...................................... 3
  2.3 Methods ...................................... 4

3 Design ............................................ 5
  3.1 Operating Systems ........................... 5
    3.1.1 Chromium OS ............................. 6
    3.1.2 Jolidoud ................................. 7
    3.1.3 Android ................................ 7
    3.1.4 iOS .................................... 7
  3.2 Virtual Machines ............................ 7
    3.2.1 Virtual Physical Memory ................ 10
  3.3 High-Level Languages ...................... 10
    3.3.1 Intermediate Language ................. 11
    3.3.2 Scripting Language ..................... 12
    3.3.3 Descriptive Languages ................. 14
  3.4 Multi-Touch Input .......................... 14
    3.4.1 Multi-Touch Input In A Linux Environment 15
  3.5 Design Result .............................. 15

4 Security ........................................ 17
  4.1 Information Assurance ...................... 17
    4.1.1 Access control ........................ 17
  4.2 Linux Kernel Hardening .................... 18
  4.3 Code Isolation ................................ 19
    4.3.1 Secure Environment .................... 19
    4.3.2 Sandboxes .............................. 20
  4.4 Execution Security In Chromium OS ........ 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Malicious Code</td>
<td>21</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Virus Detection</td>
<td>22</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Environment Detection</td>
<td>23</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Subversion</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Results of Security Study</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Implementation</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Implementation Result</td>
<td>27</td>
</tr>
<tr>
<td>5.1.1</td>
<td>An Interactive Multi-Media Application</td>
<td>27</td>
</tr>
<tr>
<td>5.1.2</td>
<td>A Web Application with Multi-Touch Input</td>
<td>28</td>
</tr>
<tr>
<td>5.1.3</td>
<td>A Bare-bone System</td>
<td>28</td>
</tr>
<tr>
<td>5.1.4</td>
<td>A Hardened System</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Future work</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Acknowledgements</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Acronyms</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>39</td>
</tr>
</tbody>
</table>
List of Figures

3.1 An illustration of virtual memory. ........................................ 6
3.2 A diagram of a type 1 VM on the left and a type 2 VM on the right. 8
3.3 An illustration of virtual memory in a virtual machine environment. 10
3.4 A flow chart of a TJIT scripting engine. .............................. 12

4.1 The program Red Pill written by Joanna Rutkowska. ............... 25

5.1 A software stack for a barebone solution. ............................ 29
Chapter 1

Introduction

Dohi Sweden is a holding company run by Emanuel Dohi that offers products and services in the audiovisual industry. They are developing a multi-touch user interface. This interface will deliver interactive content on a multi-view surface. A system is being developed that will provide a user-friendly way to populate a user interface with interactive real-time multi-media contents.

The user interface will be publicly displayed and administered by its owner. The user interface will display multiple applications with different content. These applications will contain interactive real-time multi-media content.

An OS will be selected that delivers interactive real-time multi-media content in the touch-screen user interface. What, when and how content are displayed in the user interface should be controlled by its owner with an application from a remote computer. The content will be distributed and maintained from a remote server.

In a public real-time environment, the execution and maintenance of content applications must not affect the integrity of the system. Application that executes must not affect each other nor the underlying OS. The OS must meet the above requirements and execute applications in a secure manner.

Components that can produce interactive real-time multi-media content and ensure isolated execution will be tested.
Chapter 2

Problem Description

2.1 Problem Statement

This project consist of three goals presented below.

**Design**

The first goal of this project is to identify the components needed for an OS that can deliver interactive real-time multi-media content in a user interface. The content that the OS will display will be developed by various third party companies. The target group of content developers are web designers and advertisement agencies. The OS must be updated remotely. Therefore it must be able to recover its state after a remote update or maintenance operation.

The update functionality is outside the scope of this project, although it must be considered when choosing components for the solution.

**Security**

The OS must execute content developed by third party companies regardless of its behaviour. The execution of content applications must be isolated and unable to affect each other or the underlying OS. The target group of third party developers use technologies like Flash, Silverlight and Hyper Text Markup Language (HTML)5 with JavaScript to produce interactive real-time multi-media applications. The OS will use one a high-level execution environment like the one above, but it is not decided which. It is well known that technologies like this introduce security holes in a system. Execution security will be investigated to manage possible security imperfections.

**Implementation**

The third and final goal is to test the selected components from the first goal. The components should be considered as parts of an complete OS as described in the first goal. The components must be able to deliver interactive real-time multi-media content and ensure isolated execution of content applications.

2.2 Purposes

Dohi Sweden is developing a product that requires an OS that can deliver interactive real-time multi-media content. This OS will operate on a computer connected to a back-end system via a network connection.
The back-end system will be used to administer an arbitrary number of computers, running this OS. The use of a back-end system allows for system updates and applications to be automatically distributed among several computers with less effort from the administrator. The administrator is the one that starts and stops execution of applications. The user interface of every computer is displayed publicly for anyone to use.

The interactive real-time multi-media applications executed in the OS will be constructed by third party developers. These developers are profiled to have knowledge in higher level languages and web development.

Most OS that exist today are used for personal computers, where the user both administers and utilizes the OS. If these OS are used in a public environment, the administrator would update and administer every computer individually. With a back-end system that controls all computers within the system, the administrator can have effortless control over all computers. A publicly displayed user interface is a target for misuse and sabotage. Therefore, the OS that produces this interface must also be very robust and resilient.

Components used in this product can be restricted by their license agreement. Some technologies are licensed under the General Public License (GPL) user agreement. Software licensed under GPL cannot be integrated with non-compliant license agreements. The parts used in the solution must have compatible license agreements. Dohi Sweden strives to develop its own proprietary software that can be used in their products. To achieve proprietary software with the use of open source software, licenses like Berkeley Software Distribution License (BSD) and Lesser General Public License (LGPL) can be used in some degree.

2.3 Methods

As an introduction to application execution, this report will introduce OS as an execution platform. It will be shown how native applications coexist in an OS as well as how OS can coexist inside separate virtual machines. Interpreted languages, virtual machines and virtual environment (VE) will be presented as alternative ways of application execution. Security and administrative benefits of OS running within virtual machines will be explored in contrast to a regular OS. These virtual machines are presented for purposes of system update and administration.

Existing technologies will be selected for a basis of evaluation. These technologies utilize alternative execution methods to produce interactive real-time multi-media content. Relevant properties will be presented that affects how the technologies can be used as components in the solution OS.

An in-depth study about security is done in order to design a secure solution. The study will cover aspects of OS security, VEs and high-level execution. The built-in security of operating systems will be presented as well as the methods used to prevent malicious code execution and privilege escalation. The security of the existing technologies described above will be evaluated.

The different technologies selected as components in the system solution will be tested as a basis for a real implementation.
Chapter 3

Design

To achieve a system solution an OS and a run-time platform is needed. Components for administration and back-end connection is outside the scope of this thesis. The OS and the run-time platform could be the same if native applications are executed directly. A solution like this would not satisfy the requirements for the run-time platform. The developers of third-party applications should be able to develop applications without extensive programming knowledge.

The different components in the software stack will be presented and explained. Candidate technologies will be presented for every component in the software stack. As a result of the design goal, technologies will be selected from these candidates to form a software stack that can be used for a system solution.

3.1 Operating Systems

A regular OS on a personal computer provides facilities to execute multiple applications at the same time. Applications are executed as processes on top of the OS, using time sharing. Hardware resources are distributed even among the applications.

Every application is given a window of execution time. When the window has passed, a context switch will occur that switches execution stack, file descriptors and process specific memory. The operation system also acts as a hardware abstraction layer. Applications can share hardware resources without interfering with each other.

The OS does not expose the real memory addresses to the processes. If it did, applications would write over each other’s data. Instead every process uses a virtual memory, that represents the real memory, but needs to be translated in order to get the correct data.

Virtual memory enables multiple applications to use the same physical memory [44]. The applications cannot directly access each other’s memory due to virtual addressing. The physical memory are sectioned into frames. The virtual memory has the same size and structure. The sections in the virtual memory are called pages. A page table are used to translate logical addresses to physical addresses. To aid in this translation, a hardware solution called translation look-aside buffer is used. The most frequently used pages can be stored in the TLB to be accessed with less performance degradation. If the physical memory runs out, rarely used pages are swapped out to a backing store.

The OS provide execution security with the use of virtual memory. virtual memory prohibits separate applications to use the same memory area. This allows applications to execute independently without interfering with each other.
In figure 3.1 virtual memory is illustrated for two applications executed by an OS. Application A and Application B use the same physical memory, but they address their own virtual memory. Every virtual memory address is translated into a unique physical memory address. The virtual memory is the same size as the physical memory. If two applications are executed at the same time, the virtual memory will exceed the physical memory. When the physical memory is full, some pages will be evicted to a backing store. If the physical memory does not contain the virtual memory address, it is retrieved from the backing store.

Four different operating systems will be presented. Chromium OS and Jolicloud are developed for personal computers with limited capabilities. Android and iOS are developed for embedded devices.

### 3.1.1 Chromium OS

Chromium OS is currently being developed. It is an open source project under various license. It aims to be a fast, simple and secure OS.

The user is limited to "on-line" applications provided by Google and other network-based application vendors. The user authorization is done via a Google account. The OS demands access to internet and Google servers to provide these applications.

The only unknown code that will execute are downloaded from the internet in form of web applications. web applications are discussed in section 3.3.3. Every web application that is executed on Chromium OS will do it inside a sandbox. The OS and the user content are separated with different partitions. The system is in turn separated into two copies for recovery purposes.

Chromium OS uses the Linux kernel. Chromium OS will provide custom firmware to verify that the kernel has the right signature. If the kernel has been tampered with, an external drive can be used to recover the system. During boot-up, system packages are also checked with digital signatures by the kernel. The hardening of the Linux kernel is not
yet finished [3]. Various approaches are being discussed to reduce the exposed surfaces of the system. The kernel has mandatory access control. Processes are being put in sandbox environments. The overall kernel hardening revolves around the principle of least privilege.

3.1.2 Jolicloud

Jolicloud is an OS based on the Ubuntu Netbook Edition Linux distribution. The OS is built for computers with limited disk storage and memory. Jolicloud is built to be used with internet applications. All application and system updates are performed automatically. Personal settings are synchronized automatically with an internet cloud solution. The Jolicloud user interface uses Chromium web browser. It is built on HTML5 and displayed inside the Chromium web browser.

3.1.3 Android

Android is an OS for embedded devices. It has a full software stack for mobile devices and an application framework to manage the whole system.

Applications development for the Android platform is done with the Java programming language. The Eclipse integrated development environment can be used for development. Applications are executed by the Dalvik virtual machine (VM). The standard development kit features a device emulator, debugging tools and profiling tools.

Android has integrated libraries for multi-media, structured storage, secure network access and internet protocols.

3.1.4 iOS

iOS was originally developed for the iPhone. Today it runs on iPod Touch, iPad and Apple TV as well. The iOS is a closed-source OS that only is designed to run on products from Apple Inc. It has a open source kernel named Darwin.

The iOS has integrated libraries for multi-media, structured storage, secure network access and internet protocols.

Applications development for iOS is done with the ObjectiveC programming language. The integrated development environment Xcode is used to develop applications for iOS. The standard development kit features a debugging environment, application tuner and a iOS simulator.

3.2 Virtual Machines

Popek and Goldberg defines, in their paper [33], a VM as “an efficient, isolated duplicate of a real machine”. This is called full virtualization and mimics a full set of hardware resources.

Full virtualization supports the virtualization of the processor, memory and I/O devices. The processor or Computer Processing Unit supports a given set of instructions that performs different computations. This instruction set is called an Instruction Set Architecture. In most cases the virtualized OS uses the same ISA as the underlying hardware. Partial virtualization allows certain applications to run within the VM, without the presence of an OS.

If the virtualized OS does not use the same ISA as the underlying hardware, the virtualization layer must emulate the virtualized machine ISA. This is a very inefficient method.
To enhance performance, instructions can be executed on the processor. This requires that the virtualized OS is made to use the same ISA as the underlying hardware.

In order to virtualize more than one OS on the same computer, a hypervisor is needed. The hypervisor presents a hardware interface to the virtualized operating systems. It also multiplexes resources between the virtual machines, so that all virtualized operating systems are isolated from each other. The hypervisor is also called Virtual Machine Monitor (VMM).

There are two types of virtual machines. A type 1 VM operates as an OS and a hypervisor at the same time. A type 2 VM is executed on top of a host OS. All virtualized operating systems in a type 2 VM are called guest operating systems.

![Diagram](image)

Figure 3.2: A diagram of a type 1 VM on the left and a type 2 VM on the right.

The left side of figure 3.2 shows a type 1 VM [35]. A type 1 VM has a virtualization layer that acts as an OS for the virtualized operating systems. The virtualization layer performs scheduling and resource allocation for all the virtual machines that run within the system.

To implement a VM of type 1, there are some hardware requirements [35].

- Non-privileged instructions need to be equal in both privileged and unprivileged modes.

- There must be a memory protection system.

- All sensitive instructions must be unconditionally controlled.

The right side of figure 3.2 shows a type 2 VM. The virtualization layer is executed as an application on top of the host OS. The host and guest OS both use the same hardware, but the host OS can execute privileged instructions. All hardware requirements of a type 1 VM must be met in order to implement a type 2 VM. In addition, the following requirements must be met.

- The host OS cannot change the way the guest OS execute privileged instructions.

- The host OS needs primitives to redirect sensitive instructions when they are trapped.

As mentioned above, there are two types of virtual machines. VMware vSphere Hypervisor is a type 1 VM and VirtualBox is a type 2 VM.

VMware vSphere Hypervisor is a closed-source virtualization environment. This environment enables seamless integration of applications between various operating systems. VMware is built to virtualize many operating systems at a time. It utilizes shadow paging to enhance performance.
VirtualBox is a type 2 VM. It has an open-source version and a closed-source version. VirtualBox can be used to integrate guest operating systems with a host OS.

Some virtual machines operate with the goal of security and not flexibility. A VM can also be used to confine an executable application in a sandbox environment [39]. In a sandbox the user can specify privileges for each program. If the program tries to operate outside of its privileges it will fail. The policies are upheld by “hooks” in the OS to ensure that no privileged instruction is executed without privileges.

In order to construct a VM, the memory needs to be virtualized. In the case of a system VM, the physical memory is virtualized. In a functional VM, the VM present programs with their own virtual memory. A VM can emulate the ISA or execute the code directly on the Computer Processing Unit (CPU) without emulation or interpretation. This requires that the program or OS that is being virtualized to use the same ISA as the CPU. In an ISA there exist some sensitive instructions that can be used to take control over a whole OS. These instructions must be emulated for a system VM to function. The methods described below can be used to achieve VEs.

**Trap-and-Emulate**

Trap-And-Emulate is used to control the execution of a guest OS. When the guest OS executes a privileged instruction, it fails and generates a fault that is trapped by the hypervisor. The instruction is then emulated by the hypervisor to ensure consistent behaviour by the guest OS.

This technique was not possible on x86 architectures before the ISA had hardware-assisted virtualization. Some privileged instructions could be executed outside of privileged state without generating a fault.

**Hardware-Assisted Virtualization**

Intel and AMD processors today provide support for virtualization in their ISA. Intel’s implementation is called VT-x (Vanderpool) and AMDs implementation is called AMD-V (Pacifica). These technologies provide the functionality to use Trap-And-Emulate virtualization. They introduce a privileged mode and shadow page tables to aid virtualization. Shadow page tables are used to reduce the performance costs of the address translation illustrated in figure 3.3.

**Paravirtualization**

Paravirtualization presents a pretended hardware interface that is more suitable to virtualize than the underlying hardware.

Paravirtualization also makes the guest OS aware of its presence. Some of the hardware drivers used by the guest OS might be designed to interact with a VE rather than hardware. The guest OS can thereby communicate with the host OS. This is done with a so called “hypercem” to the hypervisor.

**Binary Translation**

Binary translation is used to virtualize an OS made for a different machine architecture than the underlying machine architecture. The VM simulates the complete hardware and emulates the ISA.

The advantages gained from a VM is that applications from different vendors can operate on the same physical machine simultaneously. The applications would be unaware of each other presence. The applications might not even be constructed to function on the current hardware, but environment can be emulated. This functionality also introduces a performance degradation. Both memory and computation must be monitored.
3.2.1 Virtual Physical Memory

Applications use virtual memory that is translated to physical memory addresses by the OS. A VM does its own address translation into physical memory addresses. Memory virtualization results in two address translations for a guest OS to access the correct physical memory address.

Figure 3.3 illustrates how a type 2 VM executes two guest operating systems. Application A is executed by Virtual Machine 1 and Application B is executed by Virtual Machine 2. The host OS executes the virtual machines as applications. Every application has its own virtual memory. In turn, every VM has its own OS. The virtualized operating systems translates their applications virtual memory to their virtual physical memory.

A VM is used to contain a whole OS. The same methods can be used to isolate applications inside VEs. To produce a sandbox or jail, a VE is set up to restrict an application from the outside system.

3.3 High-Level Languages

In section 2.2 the third party application developers are described. They will develop interactive real-time multi-media applications in a high-level language. There exist several technologies that both utilizes high-level languages and can be used to produce interactive real-time multi-media content. The benefit of high-level languages are that they can be executed on any processor architecture as they are interpreted or translated during execution.

I have categorized these technologies after language type. Intermediate languages are statically typed languages that can be binary translated into machine code. Scripting lan-
3.3. High-Level Languages

guages are dynamically typed languages and cannot be directly translated into machine
code. Descriptive languages is used to describe content rather than execute instructions.

3.3.1 Intermediate Language

Virtual machines can be used to execute multiple operating systems on the same computer.
Intermediate languages can be used to execute applications on top of an OS. The intermediate
language is compiled from a high-level language such as C# or Java. They use bindings
to low-level libraries to perform common or performance critical tasks. The language is
intermediate in the way that it is not hardware interpreted, but software interpreted.

A virtual instruction set is executed by an interpreter. This reduces performance, but
the virtual instruction set is atomic instructions that can easily be translated into machine
code. Many interpreters use Ahead-Of-Time (AOT) or Just-In-Time (JIT) compilation to
reduce performance degradation.

If the program have a long runtime on the same machine it would be appropriate to
use AOT compilation. In this way the application can run with the same performance as
a program compiled for the ISA of the real machine. If the program has a short execution
time and only will run once, the JIT compilation method would be preferred. This will
interpret the application and only translate frequently used parts of the program.

Today there exists two main-stream implementations of computer intermediate lan-
guages. The Oracle Java VM [27] interprets bytecode (JVML). The Mono [12] and Microsoft
C# [43] development frameworks interprets the Ecma International (ECMA) Common Lan-
guage Infrastructure (CLI).

Both these technologies are used in platforms that deliver interactive real-time multi-
media content. JavaFX is an application programming interface based on the Java platform.
Silverlight is a product from Microsoft that utilizes C# and the .NET platform. Moonlight
is the Mono equivalent to Silverlight.

The program code itself is an intermediate language that is translated at run-time into
machine code for the specific processor architecture. The translation phase introduces a
performance penalty compared to native execution.

Common Language Infrastructure

Common Language Infrastructure (CLI) is a ECMA standard that describes a language
in which applications written in multiple high-level languages can be compiled to. The
applications can then be executed in different system environments.

Microsoft .NET is an implementation of the CLI. It is also a platform for building
applications. Common tasks are done with low-level libraries.

.NET has a VM called Common Language Runtime (CLR). Programming languages like
languages C# and VisualBasic.NET are compiled into a CLI representation. This compiled
code is executed by the CLR execution layer.

Java Virtual Machine Language

The JVML is the intellectual property of Oracle Inc. JVM is a VM that accepts a computer
intermediate language. This language is referred to as bytecode. It executes bytecode in the
form of class-files. A program can consist of many class-files. A program may be packaged
The bytecode instruction set is interpreted or compiled during runtime. The JVM has many security features that prohibits unwanted code execution. JVM has a stack-based architecture. Other languages than Java can be compiled into bytecode.

Silverlight

Silverlight is used for developing rich internet applications. Silverlight is an implementation of the .NET Framework. Silverlight applications are executed on the .NET platform.

Windows Presentation Foundation (WPF) technology is used by Silverlight. WPF is a library that contains services for user interface controls, digital rights management, graphics, text, animation and images. Silverlight uses Extensible Application Markup Language (XAML) to declare user interface objects and layout.

Mono is an open source implementation of the .NET framework. The Silverlight framework is substituted with the open source equivalence moonlight framework.

JavaFX

JavaFX is a platform for delivering rich internet applications. It is scripted with the declarative scripting language JavaFX script and runs on JVM. JavaFX script resembles both Java and JavaScript. JavaFX script uses data binding, which is a simple syntax for synchronizing the state of multiple objects.

Java libraries can also be linked in JavaFX applications. It has animation features like time-lines, key-frame animation, tweening, and path based animation.

3.3.2 Scripting Language

scripting languages are used for easy and fast development, but encourages the use of bad programming practices. Scripting languages like JavaScript and ActionScript are frequently used to produce interactive multi-media content. A scripting language is self modifying and dynamically typed [49]. The dynamic typing makes it impractical to compile it to machine language. Therefore it is often interpreted in the same way as intermediate languages. The performance degradation of the interpretation is extensive. A lot of effort has been put into the development of Just-In-Time compilation for scripting languages [7, 19, 20, 5, 21, 4].

![Flow chart of a JIT scripting engine.](Figure 3.4)
In the process of interpreting a scripting language, the language goes through a series of transformations. First, the language has to be parsed and sectioned into tokens that describe the syntax. These tokens will be analysed and parsed into a parse tree.

The parse tree is the basis of the execution. It will be the representation of the program that the interpreter executes. The interpreter traverses the parse tree and performs the instructions therein.

The interpreter collects runtime statistics from every jump in the parse tree. This is called tracing [4, 21, 5]. The runtime statistics are used to determine "hot-spots" in the code. The "hot-spots" are code segments that are executed frequently. When a "hot-spot" is identified it is compiled by the JIT compiler. Compiled "hot-spots" can be executed as native code without being interpreted. The performance is enhanced because the repeated interpretation degrades performance more than the compilation.

The dynamic execution and typing causes changes in the code during execution. This will in turn cause deprecation of the native code. The code must therefore be re-evaluated to determine if a new compilation should be made. Another method of JIT compilation is to compile functions as separate code sections.

Scripting languages are usually used to perform small tasks. Shell scripts in a Linux or Unix environment performs sequences of automated tasks. Scripts is also used for web applications. The web server might be scripted with PHP or ASP.NET. The web page that is displayed within the browser is scripted with JavaScript. The Flash technology also utilizes a scripting language to execute its applications.

**Flash**

Adobe has developed Flash, an application platform using an interpreter to deliver rich internet applications. AIR is a runtime that can be executed on multiple operating systems. The Flash technology uses AIR to execute as a desktop. Flex is a building and deployment platform for Flash and AIR.

Flash provides a platform for developing rich media applications for mobile devices, the web and the desktop. ActionScript is used to develop Flash applications. It is based on JavaScript.

**Scripting Libraries**

Several high-level programming libraries have been developed to ease the development with JavaScript and the DOM model. The most common JavaScript libraries are jQuery, Prototype and script.aculo.us. They are used to simplify HTML document traversing, event handling, animation and Asynchronous JavaScript and XML (AJAX) interactions.

**Processing.js**

Processing.js is a programming language for image manipulation, animation and interaction. It uses JavaScript to draw shapes and manipulate images on the HTML 5 canvas element.

Processing.js is an implementation off Processing. Processing is also a programming environment and application runtime that utilizes hardware accelerated rendering. It can manipulate two and three dimensional graphics, text, and images. It can be extended with libraries and has export capabilities. Processing uses a scripting language with syntax similar to Java.

\footnote{A counter will be attached to the jump destination in the parse tree.}
3.3.3 Descriptive Languages

descriptive languages is a very high level of representation. They are formal languages that describes what to do rather than how to do it. descriptive languages can be used for rapid development with specific purposes.

Examples of descriptive languages are HTML, Cascading Style Sheets (CSS), XML User Interface Language (XUL) and Open XML Paper Specification (OpenXPS). HTML describes a text document and CSS defines the style of this document. OpenXPS describes a document containing text, raster images, vector graphics and text. XUL is used to describe a different widgets in a user interface.

descriptive languages are in many cases used to present a document or a user interface. XUL is used to describe widgets and layout in Mozilla products. Similar languages with the same purpose are XAML that is used by Microsoft’s products and Macromedia eXtensible Markup Language (MXML) that is used by Adobes products.

Documents containing text, raster graphics and vector graphics in two or three dimensions can also be represented with a descriptive language. Examples of this are Scalable Vector Format (SVG) that is a The World Wide Web Consortium (W3C) standard and OpenXPS that is a ECMA standard.

A web application is an application that can be accessed by a web browser over a network, such as internet. The part of the application that is downloaded and executed on the web browser can be an applet or a web page using AJAX. The browser acts as a thin client that can interact with a back-end over the network via web services. web application use technologies like HTML, CSS and JavaScript to present interactive real-time content inside a web page. HTML, CSS and JavaScript cannot produce all multi-media content. The HTML 4 standard does not specify a canvas that can display video content.

The HTML 5 standard is specified to support graphics, video and audio. SVG images and MathML documents can also be used inside a HTML 5 document. This enables playing video in a web browser without the need of an embedded application. Before HTML 5, Flash was used to produce video inside a web page.

The HTML standard have previously supported widgets like buttons, text fields, text areas, check boxes and radio buttons. The HTML 5 standard introduces sliders, date pickers and dedicated input boxes.

3.4 Multi-Touch Input

Usual input devices for a desktop computer are keyboard, mouse. There also exist input devices like touch-screen, joystick and joystick. A touch-screen that can detect multiple touch-points on the touch-surface at the same time is called multi-touch input device. The run-time platform is required to support multi-touch input.

A multi-touch input can be represented in different ways. A common standard is the TUIO protocol. This protocol can be used to transfer multi-touch data from an input tracker application to a client application. A common way to interpret multi-touch input is to identify gestures. The gestures are triggers for user interface interaction operations. The detection and interpretation of multi-touch input events are covered in Lundgrens thesis work [28].

The Flash and Silverlight frameworks have support for multi-touch input. JavaFX does not have native support for multi-touch input, but there exist third party implementations that can produce gestures. In the HTML document object model multi-touch events is not supported. The W3C is working on a specification for multi-touch events [47] in the HTML
document object model. The Chromium, WebKit and XULRunner web browser engines have multi-touch input support of various degree, but none of them strictly follow the W3C specification.

3.4.1 Multi-Touch Input In A Linux Environment

The use of multi-touch input is not yet standardized into common operating systems. Most Linux platforms, including Chromium OS, uses X.org to gain input events from input devices. X.org drivers are used to achieve input from various devices. X.org extensions are used to deliver different input events. In order to receive input events from an X.org extension it must be supported by the application.

The X.org extension XInput2 [24] can be used to achieve multi-touch input with multiple pointers. The company Canonical has produced another multi-touch solution called utouch. Utouch uses its own framework [26] to deliver generated gestures.

The W3C defines a API [38] for receiving multi-touch input events in a web application. The events can be interpreted by JavaScript. Some web browsers such as the web browser Safari on the iPad device incorporate gesture handling for controlling the web browser.

3.5 Design Result

The problem statement in section 2.1 defines a target group of content developers that consists of web designers and advertisement agencies. The content developers should be able to develop applications for the platform with very little schooling. Developers of this type are in general used to technologies like AJAX and Flash. Compared to native applications, these technologies also offer a simple way to develop interactive real-time multi-media applications in a short amount of time. These arguments overcomes the performance penalties of high-level execution described in section 3.3.1.

The technologies that have been considered as a high-level execution platform is Adobe Flash, Microsoft Silverlight, Oracle JavaFX, Mono Moonlight and AJAX with HTML5. Adobe Flash, Microsoft Silverlight and Oracle JavaFX are proprietary software that is owned and controlled by respective corporations. Mono Silverlight is open source implementation of Microsoft Silverlight. AJAX with HTML5 is an open standard without proprietary licensing.

The proprietary software frameworks described above does not provide the freedom needed for this solution. This leaves Mono Moonlight and HTML5 with AJAX as alternatives. As mentioned above, the target group of content developers consists of web designers and advertisement agencies. This rules out Moonlight as a development framework. HTML5 with AJAX makes development of applications easy. Most interface design can be done with descriptive languages and the back-end programming with any programming language. Web pages can be used with Simple Object Access Protocol (SOAP) and database couplings to produce local as well as remote implementations. This is the reasons why HTML5 and AJAX is a much more suitable platform for Dologi Sweden as a company.

HTML5 and AJAX was selected as an application framework. It requires little effort to construct a simple application and still it is possible to develop a very large and complex application with the same framework. For many year's Adobe Flash has been a platform for delivering interactive, real-time and multi-media applications as embedded objects inside web pages. With the JavaScript execution optimizations of modern web browsers and the introduction of HTML5 it is possible to develop interactive real-time multi-media applications without embedded technologies like Adobe Flash.
The restrictions set for the OS is that it must be able to interact with the input devices developed by Dohi Sweden and execute real-time interactive multi-media applications. Many operating systems can be used in this implementation. A desktop OS is easy to set-up and will be more than sufficient to execute real-time interactive multi-media applications. Commercial operating systems like Microsoft Windows have a very large user base and security holes are discovered continuously with every release. Security holes are discovered in most operating systems, but they need to be patched in the same rate. The security of the kernel needs to be secure enough to execute any type of malicious application.

To execute real-time multi-media content can be very processor intensive. Therefore hardware acceleration of rendering operations might be needed. This will in turn require drivers for graphics cards to achieve hardware acceleration. Linux has a broad set of hardware drivers for graphics cards of different vendors. The current implementation of the Dohi Sweden’s input driver only works with the Linux kernel 2.6. This is also why other OS kernels haven’t been considered.

Chromium OS is built to run web applications. It uses a Linux kernel that have been altered to ensure as secure execution environment as possible. This makes Chromium OS a suitable candidate as an OS and run-time execution platform. Jolicloud is also a fitting candidate. The authors of Jollicloud have no public documents available to support the use of a secure design. This is why Chromium OS was selected as OS and run-time platform. Alternatively BSD or Linux can be used as OS. This requires further measures for a secure execution of third party applications.
Chapter 4

Security

This in-depth study of execution security is motivated by the fact that a publicly displayed interaction system like the one developed by Dohi Sweden could be a target for sabotage and misuse. Different Content Applications that executes in the Application Runner comes from various vendors. In order to ensure a secure execution environment for each individual Content Application, the Application Runners needs to be logically isolated from each other. If one Content Application compromised the underlying Client OS, all other Content Applications would be affected as well. Even if all Content Applications would be thoroughly examined and quality assured by Dohi Sweden, there would still be possibilities to exploit the system.

4.1 Information Assurance

Information assurance is an important part of computer security. In the USA, Department of Defence [14] defines information assurance as “Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities.”

A system must uphold the mandatory security policy with individual authorization. Every individual that utilizes the system must be identified and authorized. Auditing information must be kept so that actions affecting security can be traced to the authenticated individual. Every part of the system must also be assured to uphold the mandatory security policy. The system must also be documented with user manuals, test documentation and design documentation.

4.1.1 Access control

There are many different ways to achieve secure execution in a system. In a regular OS like Ubuntu Linux, Microsoft Window 7 and Apple MacOSX, the execution security has policies that is uphold by mechanism through access control and virtual memory.

An OS has policies that describes which user or group account that can access certain resources [44]. A group can hold many user accounts. A mechanism called reference monitor consults a policy table every time an application tries to access a resource. Access to the resource is granted if the application is authorized to access the resource in the policy table.
Regularly there are three privileged levels for resources in a Linux environment – read, write and execute.

The use of access control is the most basic type of security. The goal is to protect objects from unauthorized access or manipulation. Discretionary Access Control (DAC) is a common access control technique. DAC assigns permissions to objects. Every subject can assign policies if the current policies allow it. Mandatory Access Control (MAC) is a hierarchical access control environment. Only policy administrator can set policies.

Access control can be enforced by various methods. Access Control Lists (ACL) consist of permission lists attached to specific objects. When a subject requests access to an object, the policy is determined by the ACL. Role-Based Access Control (RBAC) is a variant of MAC and DAC. All subjects are assigned a role. If a subject is assigned a role, it is granted all the privileges that comes with the role. In capability-based security different keys are used as capabilities. A capability is a communicable, unforgeable token of authority that infer access rights to an object.

Organisation-based access control (OrBAC) is a more advanced security model that is designed for organizations. It every action a subject can do on an object is constrained by a context.

To achieve a highly secure system, a highly secure policy must be constructed. If there exist a hole in the security policy the system will be defenceless, regardless of the security model. By understanding the security model the right access policy can be set to help stop such a privilege escalation. Further complexity is added in an asynchronous system. In a real world application, problems like time-of-check-to-time-of-use becomes a greater issue when multiple processes executes simultaneously.

4.2 Linux Kernel Hardening

Chromium OS uses a Linux kernel. It is designed with its own security measures, but there are more security measures that can be taken in order to prevent privilege escalation inside the kernel.

Memory protection and access control is the main security measures taken in an OS. The Linux kernel uses Discretionary Access Control (DAC) as a default security measure. DAC defines read, write and execution access control for every object. In the Chromium OS, MAC is implemented in addition to the default DAC.

By using chroot, the file system structure can be reduced for an application. It changes the root of the file system. The application won’t be able to access files below the new root. The feature is used as part of the Chromium OS sandbox. The development of Chromium OS does not yet provide all the functionality described in its design documents. The features below are possible components used to harden the Linux kernel.

There is a couple of different kernel hardening projects that can be used. Grsecurity [41] is a collection of kernel patches that strengthen the security of the Linux kernel. Role-Based Access Control (RBAC) are added to enforce privilege policies. Grsecurity also feature least privilege protections for memory pages with the use of Pageexec [45] (PaX). This is done by setting the stack as non-writeable, heap as non-executable and by randomizing the address space layout. Some ISA have a NX (non-executable) bit, but the IA-32 architecture does not. PaX emulates the NX-bit with the supervisor bit overloading. These measures prevent buffer overflow exploitation. Like PaX, the Exec Shield project [46] aims to make executable memory non-writeable.

Linux Security Modules [51] (LSM) is a framework that allows support for further security models in the Linux kernel. LSM implements MAC in the Linux kernel. LSM is bundled
with the Linux kernel as of version 2.6. However, it does not support OS-level virtualization. In contrast to LSM DAC methods, Rule Set Based Access Control [15] (RSBAC) can be used to enhance access control in the Linux kernel.

4.3 Code Isolation

The OS have built in security as mentioned above. The kernel and the users have separate modes and a clear boundary. Unauthorized code execution can be achieved by holes in the access privileges or through flaws in software.

Buffer overflow is one flaw that can be exploited to gain execution privileges [44]. When data is written outside the bounds of a buffer, memory occupied by the executing program will be altered. This can be used to inject malicious code into the executing program. This malicious code can in turn be used to redirect return addresses or interrupts.

The more code there is, the more risk for potential security flaws. Simplicity is to strive for, in a security perspective. Saltzer and Schroeder states in their article [37] that a system should be designed with the following principles.

- The system design should be public.
- No access should be the default privilege.
- Authority should be checked continuously.
- Give no more privileges than needed.
- Security should be built into the system.
- The security scheme must be accepted by its users.

It is not uncommon to use an antiviral application to identify, stop and remove viruses. Antiviral programs use signature-based detection to identify malicious code. This requires a large database with a signature of every “virus”. If malicious code is encountered whose signature does not exist in the database, the antiviral application is unable to detect it. Antiviral applications can also detect viruses by using integrity checks on executable files.

The subject of trusted software revolves around the fact that the large software systems used today has unavoidable vulnerabilities due to the size of the system [32]. If one vulnerability is exploited inside a program that has superuser privileges, the rest of the system can be compromised. This motivates the use of an OS with support for virtualization. Virtualization would be a step towards the goal of a trusted computing base [32].

4.3.1 Secure Environment

Chen and Noble [8] argues that a virtualized environment are more secure and mobile. “The state of a VM can be saved, cloned, encrypted, moved, or restored, none of which is easy to do with physical machines.” The system log is vital in a post-attack analysis. The attacker might evade or alter the log in a regular OS. Chen and Noble propose to log enough data to replay the complete execution of the VM. The log needs to be unaltered and complete. In reality, this is hard to accomplish with a multi-core processor. It is assumed to be a single-core processor. The log can be used to replay an event caused by malicious input.

1 “We can solve any problem by introducing an extra level of indirection.” [42]
The effects of a suspicious input can be examined by cloning the guest OS. The information flow in the system is monitored and traced. Other attempts to log intrusions have also been made [16].

Wen, Zhao and Wang propose a Secure Virtual Execution Environment (SVEE) [48]. The SVEE would be a VM running within a host OS. It would reproduce the execution environment of the host OS. If the user approves of the changes made by the program inside the SVEE they can be applied to the host OS.

4.3.2 Sandboxes

A sandbox can be used to enforce code isolation. The virtual address space is divided into regions, called sandboxes. These sandboxes act as an address translation on top of virtual memory. By assigning code and data different sandboxes, the code cannot modify itself during execution. Instructions in the program that produces changes to the program counter will be tested so that the program counter lands within the boundaries of the code sandbox. If the instruction forces the program counter to point outside the sandbox, it will be ignored.

Dynamic jump instructions retrieves an address from memory that is used to change the program counter. Instructions like these must be evaluated at runtime. In order to control the execution, code can be inserted before the instruction to test the target address. The application can also be compiled with a secure compiler that produced the test code a compile-time.

This kind of control is only possible with a sandbox environment. A regular desktop OS that executes native applications does not provide the tools to produce a sandbox environment.

Virtualization have long been used to run multiple operating systems on the same server machine. A virtualization environment produces multiple virtual machines that can execute independent OS through hardware abstraction. This functionality can be used to produce a sandbox or jail. By running applications inside a VE but without a guest OS. Instead the host OS is used.

OS-Level virtualization can be used to isolate applications in a VE. There exist several VE implementations [40] for the Linux platform. The Linux-VServer [10] provides kernel level isolation. The virtual server is implemented through a combination of other security enhancing tools for Linux. The user-space environment is separated into isolated VE. Linux-VServer runs multiple Linux distributions with the same Linux kernel as the host distribution.

OpenVZ [13] is a virtualized container that runs Linux distributions on a host Linux distribution with a Linux kernel. It ensures independent execution environments without the overhead of a true hypervisor. All environments have isolated network devices, process tree and access rights.

The FreeBSD OS has a mechanism called jails [34]. This mechanism executes an isolated VE running on the host machine. The functionality is very similar to OpenVZ and Linux-VServer on Linux. Similar to FreeBSD jails, OpenBSD uses sysjail [17] to isolate execution in different VE.

Linux Containers [29] (LXC) is a user control addition to Linux that virtualizes system components. It uses mechanisms to add resource management and process isolation.
4.4 Execution Security In Chromium OS

Chromium OS is explained in chapter 3.1.1. ChromiumOS is an open source OS that is designed with security in mind. The software stack only contain the necessary parts to run web applications and maintain the system. User accounts are coupled with the Google account with a single sign-on. The goal of Chromium OS [3] is a pure capability-based system.

To secure the system boot-up process a firmware is used to verify itself and later the rest of the boot-up process. A read-only part of the firmware will verify the content of the rest of the firmware. To restore a corrupted firmware, it has a read-only backup.

The firmware is also responsible for verifying the kernel boot-up process. Every block loaded from disk is verified. To enhance performance, many blocks can be bundled into larger blocks. The bundles are in turn verified. The firmware is not available for general computer configurations. This system is not persistent toward physical manipulation neither. To ensure that a trusted kernel is loaded, its signature must be verified by the firmware. Duplicate system partitions are kept for automatic updates and system recovery.

The Chromium OS Linux kernel has been hardened by applying protection mechanisms and reducing privileges. By applying the principle of least privilege the attack surface and total amount of code exposed are minimized. Control Groups [30] are used for constraining denial of service attacks. Processes are isolated and contained with namespaces by using chroot. The CPU and memory consumption are also restricted to operate within given bounds. Namespaces [11] will be used to isolate processes and process trees from each other. A combination of POSIX file capabilities [23] and a kernel patch helps keep with kernel privilege escalation vulnerabilities. Default permissions of the user with id 0 (root) can be reduced to superuser permissions in specific process trees. The process specific isolation and limitations mentioned above are set by the Linux SUID sandbox [2] that function as a secure application launcher.

All user applications that are executed on the system is executed inside separate sandbox environments. Address space layout randomization (ASLR) is used to randomly arranging the data areas in memory. This will make it harder to identify where in memory a particular data element is stored. Stack cookies are used to detect buffer overflows. A 4 bytes long pseudo-random number is placed between the local variables and the return addresses in the stack. If a stack cookie is altered it shows that a buffer overflow has occurred.

The root partition is configured as read-only to prevent tampering. The user home directory is also restricted to disallow executable files and device nodes. Resource, process and kernel interactions are limited by a mandatory access control implementation with automated learning. Further more, accessibility to devices are filtered to avoid abuse.

A long term goal of the project is to implement sandboxes for device drivers. This would prevent applications from exploiting flaws in device drivers to gain system access. The Linux kernel module Tamooyo are considered to get a more complete access control. The Linux kernel patches in gsecurity are also considered.

4.5 Malicious Code

malicious code that penetrate systems are often viruses. Viruses can be classified in different categories. A polymorphic virus is a program that replicates itself. There also exist metamorphic viruses that mutates when it multiplies. The original program is a subset of its mutation. The next step in evolution would be a self-referencing virus that changes its signature when it multiplies. Such are the characteristics of artificial intelligence.
4.5.1 Virus Detection

A virus generally tries to avoid being detected while spreading to new places. It can be contained with the use of a sandbox environment. This will prohibit the virus from spreading. A virus also needs to be detected to be stopped and terminated. Even if a virus has been stopped, counter measures have to be taken to ensure that the infection is not repeated. If the system security is penetrated, the security hole has to be identified so that counter measures can be taken.

Turing [25] has proven that whether a program finishes executing or continues forever is undecidable.

**Theorem 4.5.1.** The halting set is not computable (Turing computable).

Rice [25] has proven that any functional property of a program is undecidable.

**Theorem 4.5.2.** Any functional property of a program is undecidable.

Chess and White [9] defines all viruses as a viral set $S$. All programs $\{p, q\} \in S$ produces one or more programs as output. $\{p, q\} \in S$ if and only if $p$ eventually produces $q$ and $q$ eventually produces $p$, either directly or through a series of steps. If there exists a program $r$ with the same properties as $p \in S$ then $r \in S$. Algorithm $A$ detects the set $S$ if and only if $A(p), \forall p \in S$ terminates. Cohen uses Turing’s undecidability proof of the Halting Problem to prove that there is no algorithm $A$ that detects the set $S$. For any virus detection algorithm $A$, there is a program $p$, which reads:

$$\text{if } A(p), \text{ then exit; else spread}$$

Chess and White states that if the viral set $|V| > 1$ for any virus it is said to be polymorphic. Polymorphic code changes syntactic properties when it is mutated, and thereby its signature. The semantic properties of the program remain the same. The following polymorphic program $p$ implements algorithm $X$.

$$\text{if } X(p) \text{ then exit, else spread}$$

In a general case there is no algorithm that can correctly detect this virus $V$, nor can it detect the loose set where another virus in set $S$ has infected the file. The following is shown by Chess and White.

$$\exists V \text{ so that } \forall A, A \text{ does not detect } V.$$

$$\forall V \text{ so that } \forall A, A \text{ does not loosely detect } V.$$

In conclusion, there exist a virus that no algorithm perfectly detects nor loosely-detects.

Viruses uses the system like a regular program. Cohen and White states that there are two approaches for behaviour detection. The first approach is to model and measure deviations in behaviour. This can reveal suspicious behaviour but it can also lead to false positives. The second approach is to detect signatures of known viruses. This is more effective but will not detect unknown viruses and it can also lead to false positives. Cohen and White also demonstration that there is no algorithm that can detect all polymorphic viruses without false positives.

One way for viruses to avoid detection is to encrypt the virus body. This will change its signature. Its weakness is that the decryption algorithm can be detected.
4.5. Malicious Code

Metamorphic code reprograms itself and thereby changes both its syntax and semantics. Metamorphic viruses avoid signature detection by changing internal structure. The mutation can be permutation of subroutines, insertion of garbage instructions or substitution of instructions. Filoli [52] uses the Chomsky classification of formal grammars to define code mutation techniques. The detection of polymorphic and metamorphic code are proven to be undecidable. The problem is reduced to the Halting problem which is itself undecidable. Filoli remarks that behaviour monitoring could lead to a potential successful detection.

Wong [50] visualizes viruses from four code generators that generates metamorphic code. Hidden Markov models (HMM) are used to represent statistical properties of these virus variants. Wong trains the model to determine whether a given program is similar to a virus in the HMM. All viruses in Wong’s test were detected by similarity index.

An even larger set of viruses would be a self-Referencing program. A self-referencing program is defined by Case and Moelius III [6]. “In computability theory, program self-reference is formalized by the not-necessarily-constructive form of Kleene’s Recursion Theorem (KRT). In a programming system in which KRT holds, for any preassigned, algorithmic task, there exists a program that, in a sense, creates a copy of itself, and then performs that task on the self-copy.”

4.5.2 Environment Detection

Some viruses use anti-analysis mechanisms to avoid detection [31]. Viruses use different methods to change its signature. This will allow them to avoid detection. Some viruses try to detect whether they are executed inside a contained environment. If they are executed inside a contained environment, they might change behaviour. A virus can try controlling the environment or lay dormant to avoid detection. If the virus is unable to detect the sandbox environment, its behaviour will be the same. The virus can then be incubated and detected.

Ferrie states in his article [18] that “The simplest attack that malicious code can perform on a VM emulator is to detect it”. Ferrie also states that “a harsher attack that malicious code can perform against a VM emulator is the denial-of-service” and “the most interesting attack that malicious code can perform against a VM emulator is to escape from its protected environment”. If a security flaw in the execution environment is known it can be used by viruses to escape the environment. Ferrie separates the different virtual machines into categories with subcategories.

- Hardware-bound (paravirtualization)
  - Hardware-assisted
  - Reduced privilege guest
- Pure software (emulation)

Ferrie mentions that all instructions that the guest OS cannot execute, should be emulated. This would allow for nested virtual machines. Emulation also cloaks the host OS, although it does not hide the host OS. External timing of identical operations can be used to detect a delay in the guest OS [18].

Known mechanisms in the hypervisor can also be used to detect time differences. Peter states that “if the TLBs are explicitly flushed, then the time to access a new page can be determined by reading the time stamp counter before and after the access. This duration can be averaged over the number of TLBs to be filled.”
Known vulnerabilities of the underlying system can make a surface for attack. Also known properties of the VM can be used to detect it. For example, the CPUID instruction is intercepted by VirtualPC and altered.

4.5.3 Subversion

Modern x86 processor provides extensions to the ISA that can be used to improve performance of virtualization. However, the x86 processor architecture is not fully virtualizable. Some privileged instructions behave differently when run in an unprivileged mode. Some unprivileged instructions may even access the privileged state.

Virtual machines overcomes the x86 architecture limitations with the use of buffered code emulation [18]. Sensitive instructions in the ISA needs to be translated in order to avoid unwanted execution outside the VM. Control flow, addressing and privileged instructions are dynamically translated “on-the-fly” [1]. Input is the full x86 instruction set, and output is a safe subset without privileged instructions.

Blue Pill

The same instructions used to improve performance of virtualization can be used to subvert an OS. This was shown by Joanna Rutkowska, who subverted Microsoft Windows Vista 64 with the virus called Blue Pill [36]. The Blue Pill malware is specific for that hardware and software configuration.

In the Vista OS, all kernel mode drivers must be signed. The first step taken by Blue Pill is to load unsigned code into the Vista Beta 2 kernel. This is done by forcing a kernel driver into the pagefile by allocating much memory. The pagefile can be accessed and altered by a regular user. This allows the malware to be planted into the pagefile. When the driver is executed, so is the malware. Blue Pill utilizes the ISA extension AMD-V to subvert the Vista OS. It acts as a VM hypervisor to control the OS.

In order to avoid detection, all sensitive instructions needs to be trapped and emulated by Blue Pill. The Blue Pill can be detected by external timing. The trapped instructions can be repeatedly called in order to measure their execution time. The time can later be compared with an external measurement of another computer. Timing is also mentioned in section 4.5.2.

Red Pill

There exist a short program called Red Pill that is said to detect the presence of a hypervisor.

The Red Pill program in figure 4.1 will reveal if the current OS is under hypervisor control. The SIDT instruction is called. This instruction gives the address of the interrupt descriptor table register (IDTR). The SIDT instruction can be executed in non privileged mode and it doesn’t generate exception. Thereby the instruction is not hooked by a hypervisor, nor unreachable by a program executing on a hypervised OS. Specific virtualization software use different addresses for their IDTR and can thereby be identified. However, the result is inconclusive on a multi-core machine.

\footnote{http://invisiblethings.org/tools/redpill.c}

\footnote{In the movie “The Matrix”, the main character Neo is offered a red and a blue pill. As he swallows the red pill he awakens in the real world and discovers that he has been living inside a computer simulated world.}
4.6. Results of Security Study

In a custom system like the one developed, its security is very likely to be tested by custom viruses. This rules out signature detection as a security measure.

Privilege escalation can be stopped by the use of an appropriate security model and a solid access policy. The access policy should be set with the principle of least privilege. The most common security threats caused by buffer overflow and code injection can be stopped with additional access control of the stack and heap. This is done by setting the stack as non-writeable and the heap as non-executable. However, a non-writeable stack and a heap non-executable makes Just-In-Time compilation impossible.

A sandbox can be used to separate the execution environment of every application. This would allow every application to execute without interfering with each other. Faulty applications can be contained as well as malicious software. Viruses that use anti-analysis mechanisms to avoid detection poses little threat in the sense that they will not spread to avoid detection.

It does not seem to exist any deterministic detection methods for metamorphic viruses. Viruses like blue pill that utilizes virtualization technology built into the ISA also pose as a real threat. A detection based on buffered code emulation seem to be possible but it has not been implemented.

The system log is vital in a post-attack analysis as it can be used to backtrack viruses. To maintain an unaltered system log is vital.

It is also very hard to protect a system from physical access with purely-software. If the hardware is altered the system may be compromized. The Blue Pill virus explicitly flushes the TLB to gain write access to signed content. Attacks where the hardware is targeted seems to be hard to protect against.

Figure 4.1: The program Red Pill written by Joanna Rutkowska.

/* VMM detector, based on SIMD trick
 * written by joanna at invisiblethings.org
 * should compile and run on any Intel based OS
 * http://invisiblethings.org*/

#include <stdio.h>
int main () {
    unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\x0c3";
    *((unsigned*)&rpill[3]) = (unsigned)m;
    ((void(*)())&rpill());

    printf ("idt base: %#x\n", *((unsigned*)&m[2]));
    if (m[5] > 0xd0) printf ("Inside Matrix!\n", m[5]);
    else printf ("Not in Matrix.\n");
    return 0;
}
Chapter 5

Implementation

The system that is being designed should be able to execute interactive real-time multimedia applications in a secure environment. All applications executed on the system must not be able to disrupt or interfere with each other nor the underlying OS. A multi-media application should be able to display a combination of images, animation, audio, video, formatted text and possible 3D content. The system must be able to process input fast enough to produce real-time interaction with multi-media content with minimal delay. The interactive part of the system will require that the application framework supports multi-touch input. Experimental input methods that is explored today might be used in future implementations of the system.

5.1 Implementation Result

In section 2.1 the problem statement describes three goals. The third goal of this project is to test components needed to produce a real system. The system needs to ensure isolated execution of content applications. The technologies presented in this report can be used to implement such a system.

The goal can be separated into two parts. The first part is to deliver interactive real-time multi-media content. The second part is to ensure isolated execution of content applications. A bare-bone system would be a system that fulfils the first part of the goal. Such a system is described in section 5.1.3. To fulfil the second part, the barebone system must be hardened. It has been attempted to produce a hardened system that can produce interactive real-time multi-media content. This attempted is explained in section 5.1.4.

Web applications has been selected as a form of interactive real-time multi-media applications. In section 5.1.1 a web application implementation is described. Its purpose is to demonstrate interactive multi-media content with HTML5, CSS3 and JavaScript. In section 5.1.2, a separate implementation is described to show that multi-touch can be used to control interactive real-time multi-media content. This implementation is produced with the same technologies.

5.1.1 An Interactive Multi-Media Application

Text, animation, images, sound and video are multi-media components. A web application was produced that can deliver all of these components. The application is developed with
HTML5, CSS3 and JavaScript. It consists of five pages; Home, Map, Document, Canvas and Film. Each of the six pages demonstrates different features.

The interface has been constructed to demonstrate different multi-media features. The content area have rounded corners and a shadow. The interface text also has a shadow. A gradient fills the background of the screen.

Every page has a content area to show a different feature. Five image icons are used to navigate between the pages. Once the mouse pointer is placed above one of the icons an affine transformation will move and slightly rotate the icon to indicate that it has been focused.

The page “Home” displays an image of the Dohi Sweden company motto. The page “Map” uses Google Maps to display the position of the Dohi Sweden office. A bouncing marker indicates the position on the map. The map can be scrolled and zoomed by using six control buttons in the map area. The page “Document” shows a small document with bogus text. This document demonstrates the use of a multi-column layout and embedded images. The page “Canvas” uses the HTML5 canvas element to draw an interactive real-time animation. A fading Cubic Bézier curves are drawn around a single point. For every step in the animation, the curve is changed and the rest of the canvas is faded to white. The animation will change whenever the mouse pointer is located inside the canvas. The page “Film” uses the HTML5 video element to play a short video.

The application can be installed [22] as an application inside the Chromium browser. If it is made available on-line and registered in the Google App Store it can be integrated as an on-line application.

5.1.2 A Web Application with Multi-Touch Input

JavaScript can be used to interpret multi-touch input events in web pages. An application was constructed to demonstrate this feature. The application consist of ten boxes. Each one of these boxes can be moved simultaneously to any position on the screen.

If a pointer is located and pressed above a box, a touch event triggers the JavaScript code to alter the style sheet. As the style sheet is updated, so is the representation of the boxes.

5.1.3 A Bare-bone System

A barebone system that can deliver interactive real-time multi-media content in a user interface. Such a system can be produced as a proof of concept. It will not incorporate any additional security features. It requires a regular desktop OS with a run-time application platform installed.

In figure 5.1 a software solution stack for a barbone solution is illustrated. A real-world example of such a system could be a desktop computer. The configuration below is existing components describing a real-time multi-media solution. The hardware is a desktop computer with a multi-touch input device. The OS is Ubuntu Linux 10.10. The custom Chromium web browser will act as a run-time Platform. multi-touch input events will be delivered directly to the Chromium web browser via a custom event generator. The content application can be a web page of any kind. The web page will deliver interactive real-time multi-media content that has a front-end interface implemented with AJAX and HTML5.
5.1.4 A Hardened System

Chromium OS was selected as a hardened OS and a platform to deliver interactive real-time multi-media applications in the form of web applications. It is a multi-license open source OS that is designed with the principle of least privilege. System hardening has been applied in multiple levels by its developers. The applied security measures are mentioned in section 4.4.

Chromium OS uses a Ubuntu Linux kernel as a base. X.org is used as a windowing system to run graphical applications. X.org does not yet fully support multi-touch input. There are many ways to interpret multi-touch input into gestures. These methods are discussed in section 3.4.1. The Chromium web browser is used in Chromium OS as a user interface. The attempt made is to bundle the Chromium web browser with a module that generates multi-touch gestures. The gestures where to handle events that changed page navigation and zoom. Furthermore, it would generate touch events that could be interpreted with JavaScript. In this way web applications can be controlled with solely multi-touch input.

The attempt to use Chromium OS as base failed due to the complexity of the system and the limited time of this project. The use of WebKit or XULRunner as a run-time platform was considered, but never pursued due to the lack of time. The Linux kernel hardening that has been made in Chromium OS can also be achieved with various other open source projects described in section 4.2.
Chapter 6

Conclusions

There exists many frameworks that can deliver interactive real-time multi-media content. The common platforms and frameworks that are presented in section 3 can be used. The configurations of software solutions are many and requires little effort to combine into a working system.

Web pages were chosen as a framework for developing applications. The W3C standards of HTML5, CSS3 and JavaScript requires little effort to develop interactive real-time multimedia applications. It is also well known technologies for the targeted developers.

The complexity of this system is to achieve trusted execution of custom applications. The execution will be automated without supervision from an administrator. A malicious application or malicious application content may penetrate the system without human detection.

It is impossible to construct a totally secure system. A system can always be broken with physical access or even system knowledge. It is likewise a very complex operation to achieve escalated privileges through security holes. The security of this automated system must be constructed so that privilege escalation is hard to achieve. The theoretical methods for a program to achieve privileged escalation is to explore faults in the system. Common faults are buffer overflow and flawed access policies.

Nested barriers of isolation are constructed to mitigate flaws in both kernel and user applications. Kernel memory is isolates from user space memory. Modern operating systems use virtual memory. This isolates memory from different applications from each other. Access privileges are restricted so that only authorized can access certain objects. Some operating systems use further isolation between the program stack, heap and code space to prevent buffer overflows to enter code space.

A way to gain access to a system is to force evict memory to a backing store with lower access restrictions. System penetration can even go as far as the hardware. The firmware stored inside hardware can be reprogrammed to execute malicious code. The ISA of processors with virtualization extensions can be exploited to gain access to submerge the OS. A secure computing base cannot be achieved without a secure hardware.

In this particular system the Linux kernel is used. Many of the kernel enhancements available can be used to produce barriers of defence to coop with flawed software. User applications are also isolated from each other by virtual memory, access policies and a sandbox environment.

Security issues related to the authorization of users are not covered. This is because the system will be controlled via a network. Verification and authorization of both applications
and network connections are implicit security measures of the system. However, it is outside the scope of this thesis.
Chapter 7

Future work

The system designed in this thesis is not yet implemented. Many components are missing to integrate the system into a larger solution. A scheduler and window manager is needed to control applications. System wide security policies needs to be established. Applications, content and network connections need to authenticated and authorized before they are put to use.

The software stack needs to be evaluated pre-emptive so that security holes can be avoided. The current sandbox model is not tested against hardware related attacks such as Red Pill. Physical manipulation of the system hardware can lead to a system penetration. Even though the hardware cannot sustain a physical attack it can be used to authenticate software. The possibility of hardware authentication of certificates should be explored.

It could also be possible to use FreeBSD as OS. FreeBSD jails mechanism provides a default security layer. It is not known if all system dependencies can be run on the BSD platform.

In an interactive real-time multi-media application the human interface methods are evolving. Today's technologies like multi-touch displays and full-body gestures are not fully standardized. This system should be able to function both with the current and future input methods. Tangible and 3D interactions have been demonstrated in various projects.

Like the Chromium OS project, this will be an evolving system that needs to adapt as the technology and security threats change over time.
Chapter 8

Acknowledgements

It has been fun working with all the great people at Dohi Sweden. I have learned a lot and I wish them all the best. I would like to thank Thomas Hellström for telling me what to do, Emanuel Dohi for telling me what not to do and Mikael Jonsén for all his support.
Glossary

back-end The data processing components. 3–5, 14, 15

bare-bone A minimum amount of components. 27

boot-up a sequence of instructions that starts the operating system. 6, 21

buffer overflow A program overruns a buffer’s boundary and overwrites adjacent memory. 18, 19, 21, 25, 31

descriptive language A meta language that describes functionality. 10, 13–15

firmware A program that controls a device. 6, 21, 31

hardening The process of securing a system by reducing its surface of vulnerability. 6, 18, 28, 29

hypervisor A hypervisor or virtual machine monitor has control over guest operating systems executed on a virtual machine. 8, 9, 20, 23, 24

interactive A object that responds to stimulus. i, 1, 3, 4, 10–12, 14, 15, 27, 28, 31, 33

intermediate language The language of an abstract machine. 10–12

interpreted language A programming language which is indirectly executed by an interpreter program. 4

kernel The central component of a operating systems. 6, 7, 15, 16, 18–21, 24, 29, 31

malicious code A software designed to secretly access a computer system without the owner’s informed consent. 4, 19, 21, 23, 31

multi-core Multiple cores that execute in parallel. 19, 24

multi-media media and content that uses a combination of different content forms. Multimedia includes a combination of text, audio, still images, animation, video, and interactivity content forms. i, 1, 3, 4, 7, 10–12, 14–16, 27, 28, 31, 33

multi-touch A surface for user interaction. Multiple fingers can be used to control the device. i, 1, 14, 15, 27–29, 33

physical memory Memory, directly accessible to the CPU. 5, 9, 10
run-time  The time during which a program is executing. 5, 11, 14, 16, 28, 29

sandbox  A security mechanism for separating running programs. 6, 8, 10, 18, 20, 21, 23, 25, 31, 33

scripting language  A programming language with dynamic typing. 10, 12, 13

security  Protection against danger, damage, loss, and criminal activity. 1, 3-5, 8, 11, 15, 17-21, 23-25, 28, 31, 33

security holes  A weakness which allows an attacker to reduce a system’s information assurance. 3

software stack  A set of software subsystems or components needed to deliver a fully functional solution. 5, 7, 20, 33

Trap-And-Emulate  The virtualization layer denies access for the guest OS to the actual page table entries. The access is trapped and emulated in software. 9

Turing computable  Can be computed by a Turing machine. 22

undecidable  A decision problem which no algorithm can decide. 22

user-space  User space is the memory area where all user mode applications operate. 20

virtual memory  Physical memory, divide in a virtual address space. 5, 9, 10, 17, 20, 31

virtualization  The creation of a software version of other software or hardware. 7-10, 18-20, 24, 25, 31

web application  An application that is accessed over a network and hosted in a browser-controlled environment. 6, 13-16, 20, 27-29

x86  a family of instruction set architectures based on the Intel 8086 CPU. 9, 24
Acronyms

AJAX Asynchronous JavaScript and XML. 13-15, 28
AOT Ahead-Of-Time. 11
BSD Berkeley Software Distribution License. 4
CPU Computer Processing Unit. 9, 21
CSS Cascading Style Sheets. 14, 27, 31
ECMA Ecma International. 11, 14
GPL General Public License. 4
HTML Hyper Text Markup Language. 3, 7, 13-15, 27, 28, 31
JIT Just-In-Time. 11, 13
LGPL Lesser General Public License. 4
MXML Macromedia eXtensible Markup Language. 14
OpenXPS Open XML Paper Specification. 14
OS operating system. i, 1, 3-11, 15-20, 23, 24, 27, 28, 31, 33
SOAP Simple Object Access Protocol. 15
VE virtual environment. 4, 9, 10, 20
VM virtual machine. 7-11, 19, 20, 23, 24
VMM Virtual Machine Monitor. 8
W3C The World Wide Web Consortium. 14, 15, 31
XAML Extensible Application Markup Language. 12, 14
XUL XML User Interface Language. 14
References


REFERENCES


