

# Diploma Thesis

University of Applied Sciences Furtwangen, Germany  
Faculty of Computer Science - Computer Networking

---

## SERVER-BASED VIRUS-PROTECTION ON UNIX/LINUX

by  
Rainer Link  
<mail@rainer-link.de>

Advisor: Prof. Hannelore Frank

Advisor: Prof. Dr. Rainer Mueller

Finished: May, 28 2003

Public Release: August, 2003



# Preface

## Abstract

Evaluation and development of server-based anti-virus solutions, running on Linux/Unix, using the Internet Content Adaption Protocol (ICAP). The diploma thesis covers proof-of-concept solutions for web proxy (Squid), eMail server (sendmail/postfix) and file server (Samba), with focus on the latter one aiming to provide a (fully-featured) product.

## Motivation

On 07/21/1999, I sent the first patch to the maintainer of the AMaViS project (A Mail Virus Scanner, <http://www.amavis.org/>, GPL'ed<sup>1</sup>) fixing the AntiViral Toolkit Pro/Linux call. Since then - among other stuff - I wrote and maintained several anti-virus modules (and still do). So, with the help of other people, AMaViS supports a wide range of anti-virus products. But wouldn't it be easier to maintain only one anti-virus module, implementing a common protocol, to support all those anti-virus scanners?

Also, back in 1999, I was looking for an on-access virus scanning solution for Samba file servers<sup>2</sup>, receiving a first Linux kernel-based solution via email in June '99. More than a year later, I came across the Samba Virtual File System (VFS)<sup>3</sup>. A half year later, I dugged into the Samba VFS and started to work on a small piece of code which eventually became the samba-vscan project: on-access file scanning directly integrated into Samba (GPL'ed, too).

As nearly all the code I wrote past years was put under an Open Source License, I decided to release this thesis under the terms of the GNU Free Documentation License.

---

<sup>1</sup>GNU General Public License, see <http://www.gnu.org/copyleft/gpl.html>

<sup>2</sup>see e.g. <http://www.geocrawler.com/archives/3/281/1999/4/0/1652065/>

<sup>3</sup>see e.g. [http://sourceforge.net/mailarchive/forum.php?thread\\_id=219140&forum\\_id=4829](http://sourceforge.net/mailarchive/forum.php?thread_id=219140&forum_id=4829)

## Overview of the Thesis

**Chapter 1** gives an overview of computer-viruses and some other types of malware. As well as anti-virus technologies and anti-virus deployment.

**Chapter 2** explains possible means to integrate third party anti-virus scanners into scripts and programs.

**Chapter 3** discusses the Internet Content Adaption Protocol (ICAP) with the focus to use this protocol for an anti-virus service. The developed "icap-client" utility for scanning any file on disk using an ICAP anti-virus facility will be dissected, too. The results of some performance testings will be discussed as well.

**Chapter 4** explains briefly the use of AMaViS for protecting the mail server and the ICAP integration.

**Chapter 5** shows two possible concepts for on-access, real-time scanning of Samba shares; focused on the direct Samba integration as implemented by the samba-vscan project. Results of file retrieval tests illustrates impacts on performance.

**Chapter 6** discusses concepts for protecting HTTP/FTP transfers.

**Chapter 7** summerizes the results and gives a short future outlook.

## Credits

First of all, I'd like to thank my advisors Prof. Hannelore Frank and Prof. Dr. Rainer Mueller for their support, feedback and suggestions.

A professional thank you goes to the following persons and/or companies:

- SuSE Linux AG for funding this diploma thesis and my AMaViS work for three years.
- Travis Priest, Rui Ataide (Symantec USA) and Gerald Maronde (Symantec Germany) for providing me with the latest Symantec AntiVirus Engine product before it was public available and for various ICAP/Symantec AntiVirus Scan Engine related discussions.
- Martin Stecher (WebWasher AG) for some email exchange about ICAP and WebWasher; Oxana Herzog and Elka Plattmann for sending a special trial evaluation key for the WebWasher CSM suite.
- Christian Hofmann of DATSEC for offering the latest Kaspersky AntiVirus for File servers and a one year license key.

## Feedback et al

Please send feedback, corrections, suggestions or even flames to

Rainer Link <[mail@rainer-link.de](mailto:mail@rainer-link.de)>

I plan to maintain this thesis and release updated versions once in a while.

## History

- 1.00 - final version (May, 28 2003); non-public
- 1.01 - changed title page, added history, added Appendix A (GNU FDL).  
released to public
- 1.01a - corrected the vfs options setting, thanks for Stefan Metzmacher  
for the report (August, 9 2003)

## License

This document is licensed under the terms of the GNU Free Documentation License (see <http://www.fsf.org/licenses/fdl.html>).

Copyright (c) 2002-2003 Rainer Link, OpenAntiVirus.org.  
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant sections "History" and "Credits; no Front-Cover Texts, and the Back-Cover Text "Diploma Thesis by Rainer Link. Published by OpenAntiVirus.org". A copy of the license is included in the section entitled "GNU Free Documentation License".



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Computer Viruses and Malware . . . . .	1
1.1.1	Introduction & Definition . . . . .	2
1.1.1.1	The Infection Mechanism . . . . .	2
1.1.1.2	The Trigger . . . . .	2
1.1.1.3	The Payload . . . . .	2
1.1.2	Classification of Computer Viruses . . . . .	3
1.1.2.1	Type of Host Victim . . . . .	3
1.1.2.2	Type of (Infection) Technique . . . . .	4
1.1.2.3	Special Virus Features . . . . .	5
1.1.3	Macro & Script Viruses . . . . .	6
1.1.3.1	Macro viruses . . . . .	6
1.1.3.2	Scripting Viruses . . . . .	8
1.1.4	Worms . . . . .	8
1.2	Anti-Virus Technologies . . . . .	9
1.2.1	Scanner . . . . .	9
1.2.2	Integrity Checker . . . . .	11
1.2.3	Behaviour Blocker . . . . .	11
1.3	Anti-Virus Strategy . . . . .	11
<b>2</b>	<b>Server-based Virus Protection on Unix/Linux</b>	<b>13</b>
2.1	Requirements & Aims . . . . .	13
2.2	Integration of Anti-Virus Products . . . . .	14
2.2.1	Command line Scanner . . . . .	14
2.2.2	Application Programming Interface . . . . .	15
2.2.3	Client-server Communication . . . . .	17
2.2.3.1	Proprietary Protocols . . . . .	18
2.2.3.2	Generalized Frameworks . . . . .	20
<b>3</b>	<b>ICAP</b>	<b>23</b>
3.1	Overview . . . . .	23
3.1.1	Introduction . . . . .	23
3.1.2	Architecture . . . . .	25
3.1.3	ICAP-enabled Anti-Virus Solutions . . . . .	27
3.1.3.1	Symantec AntiVirus Scan Engine . . . . .	27

3.1.3.2	WebWasher Content Security Management (CSM) Suite . . . . .	28
3.2	Technical Specification . . . . .	29
3.2.1	Overview . . . . .	29
3.2.2	ICAP request . . . . .	30
3.2.3	ICAP response . . . . .	32
3.2.4	Performance Consideration . . . . .	33
3.2.5	ICAP extensions . . . . .	34
3.3	Example ICAP Client Implementation . . . . .	35
3.3.1	Usage of "icap-client" . . . . .	35
3.3.2	Client Implementation Details . . . . .	37
3.4	Performance . . . . .	39
3.4.1	Hardware . . . . .	39
3.4.2	Results . . . . .	40
3.5	Conclusion . . . . .	46
<b>4</b>	<b>Mailserver - AMaViS</b>	<b>49</b>
4.1	MTA-Integration . . . . .	49
4.1.1	Sendmail . . . . .	49
4.1.2	Postfix . . . . .	51
4.2	Features . . . . .	52
4.3	Conclusion . . . . .	53
<b>5</b>	<b>Fileserver - samba-vscan</b>	<b>55</b>
5.1	Concepts . . . . .	55
5.1.1	Not Integrated into Samba . . . . .	55
5.1.2	Integrated into Samba . . . . .	57
5.1.3	Pros & Cons . . . . .	58
5.2	Requirements . . . . .	58
5.3	Performance . . . . .	65
5.4	Conclusion . . . . .	69
<b>6</b>	<b>FTP-/Web-Transfer - squid-icap</b>	<b>71</b>
6.1	Concepts . . . . .	71
6.1.1	Apache as Proxy . . . . .	71
6.1.2	Squid as Proxy . . . . .	71
6.2	Performance . . . . .	73
6.3	Conclusion . . . . .	77
<b>7</b>	<b>Résumé</b>	<b>79</b>
<b>A</b>	<b>GNU Free Documentation License</b>	<b>91</b>

# Chapter 1

## Introduction

This chapter gives an overview about computer-viruses and anti-virus techniques. If the reader is interested in a particular topic, the given reference(s) are worth a reading<sup>1</sup>.

### 1.1 Computer Viruses and Malware

The term "computer virus" was first applied to self-reproducing computer programs by Len Adelman back in 1983. One year later, Fred Cohen "scientifically defined the term computer virus" ([EK2001, p. 6]):

"We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorisations of every user using it to infect their programs. Every program that gets infected may also act as a virus and thus the infection grows" ([FC1984]).

So, in short a virus is a program which is able to replicate with little or no user intervention, and the replicated program(s) are able to replicate further. Like its biological counterpart it needs a host. In general, a computer virus is platform-dependent, i.e. a virus written for MS-DOS will not run under Linux/Unix (but it may run of course under DOS emulators like DosEmu or "Virtual PC systems" like VMWare). One exception are macro viruses (see below) or Java viruses (like BeanHive, see [CR1999, pp. 9]); and some few examples of computer viruses written for Windows and Linux, like Lindose aka Winux ([JK2001, p. 150]). The first virus for Apple II was the Elk cloner virus back in 1981. In 1986, the first virus for the IBM PCs and compatibles appeared: the Brain (aka Pakistani or Ashar) virus ([EK2001, p. 6], [AM2000]). [EK2001], [FP2000] or [AM2000] cover history in depth.

---

<sup>1</sup>If possible, more than one reference has been mentioned. A bibliography reference without a page reference covers a topic solely and/or in depth.

### 1.1.1 Introduction & Definition

Generally speaking, a computer virus consists of three parts ([MR1995]):

- the infection mechanism,
- the trigger,
- the payload.

As mentioned above, a computer virus must at least have the infection mechanism part.

#### 1.1.1.1 The Infection Mechanism

As the name already implies the **infection mechanism** ([CS1995, p. 10]) searches for one or more suitable victims and checks to avoid multiple infections if the host is already infected or not (not every virus does this; some viruses infect a host multiple times due to bugs). After that, simply speaking, the virus body is copied into the victim. The easiest method to do so is (by) overwriting the code of the victim. Other methods are putting the code in front of or at the end of a file.

#### 1.1.1.2 The Trigger

A **trigger** ([CS1995, p. 10]) is used for starting the possible payload, i.e. on a particular event, the payload is executed. Such an event could be a special day (Friday, 13th) or when the infection counter has reached a pre-defined value.

#### 1.1.1.3 The Payload

```

MOSAIC   BAK      10691 11.11.93   15.32
MOSAIC   INI      10683 20.01.94   19.50
APPLICA@ GRP      4693 23.01.94   15.33

```



Figure 1.1: Payload of Ambulance Car virus, taken from [FSC2003]

A possible **payload** ([CS1995, p. 8]) causes transient or permanent damage, e.g. displaying an animation on the screen (e.g. a red cross car moves along the screen, see figure 1.1); or formatting the hard disk drive or manipulation of data.

Of course, damage may even happen unintentionally, e.g. due to a programming error or if an old DOS virus causes trouble within the Windows environment (see e.g. [MO1997]). Damage may be caused by over-reaction by the user ([BSI1994, p. 1-8]), too.

### 1.1.2 Classification of Computer Viruses

The **classification of computer viruses** ([MR1995]) can be done via several ways:

- type of host victim,
- type of infection technique,
- special virus features.

#### 1.1.2.1 Type of Host Victim

As of **type of host victim** ([MR1995]) we can distinguish between

- boot (DBR) sector and master boot record (MBR) virus,
- file virus,
- companion virus,
- multipartite virus.

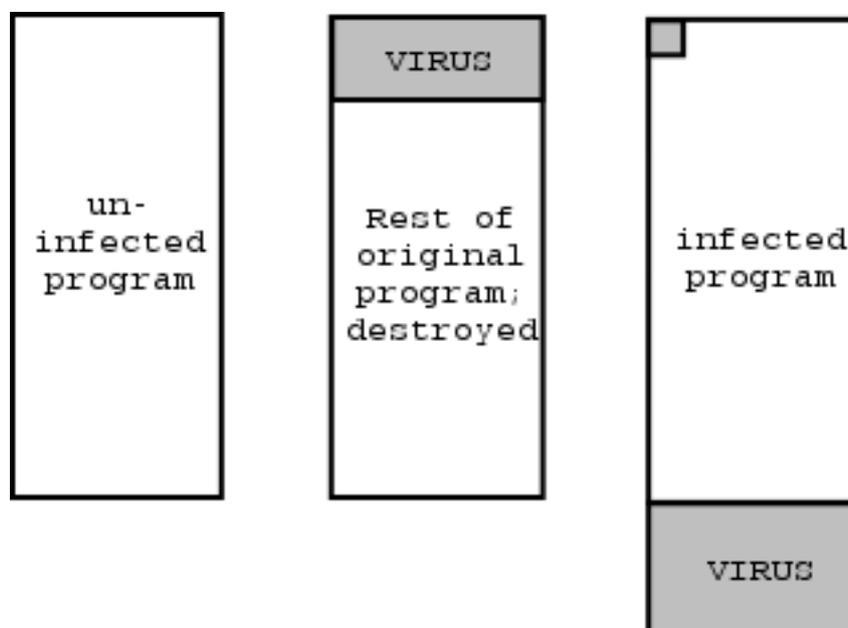


Figure 1.2: Overwriting / Appending virus

**A boot virus** infects the boot sector of a floppy disc and/or master boot record or boot sector of a hard disc. Such a virus can infect the computer system, when the computer is booted from an infected floppy disk. As the code in the MBR/DBR is started by the BIOS after it does the POST (Power On Self Test), the virus gets activated even before the Operation System has been started and most likely "hooks" some particular Interrupts (e.g. BIOS INT13h or DOS INT 21h) for performing its tasks. Most boot sector viruses are memory-resident, so they can easily infect every non-write protected floppy when it is accessed. Most viruses of this type save a copy of the original boot sector/master boot record in an unused sector of the disk. A boot virus may be "placed" into the computer system by a so-called "dropper", i.e. a program which simply drops the boot virus.

**A file virus** infects (executable) files, either by overwriting the file (overwriting virus) or by appending the virus code at the beginning or end of the file (appending virus). An overwriting virus destroys the original file upon infection. Most appending viruses put their virus code at the end of the file and put a jump to the virus code at the beginning of the file, so that the virus code is started first upon execution.

**A companion virus** looks for programs with the extension .BAT or .EXE and then creates a .COM file with the same name (i.e. TETRIS.COM, if a program TETRIS.EXE exists). If only the program name is entered (here: TETRIS), DOS per default looks up first for a matching .COM, .EXE and then .BAT file. So, TETRIS.COM will be started (instead of TETRIS.EXE, which was originally the intention of the user). Therefore, the companion virus is started first and can then start TETRIS.EXE ([RS1996, p. 24-25]).

**A multipartite or hybrid virus** uses more than one infection technique, e.g. a combination of a boot sector and file virus and therefore infects DBR / MBR and files. Or viruses which infect Office files via Visual Basic for Applications (VBA) and Visual Basic Script (VBS) files (see section macro viruses, p. 6); or viruses which infect Win32 files and office files, like Win32/W97M.Beast ([PS1999, pp. 6]).

The basic infection technique of file viruses for Windows systems are somewhat similar to DOS viruses (see above), but (much) more complicated as the file format is more complex, too ([PS1998], [PS2000]). This applies basically to Linux viruses ([JK2000], [MVO2000]). Even viruses for both platforms are possible ([JK2001]).

### 1.1.2.2 Type of (Infection) Technique

Basically, the technique ([MR1995]) can be distinguished between

- direct action (non-memory resident),

- memory resident.

**A direct action virus** ([MR1995]) does not stay in memory, so it's only active when an infected program has been started and only by this event it can replicate. A direct action virus is not very complex and can therefore be very small (the Trivial-31 virus is only 31 bytes "big"). In most cases, a direct action virus does not spread as fast as a memory resident virus.

**A memory resident virus** ([MR1995]) installs itself into RAM and may be active as long as the computer is running. This can be achieved via several ways, depending on the operation system: DOS provides a mechanism called "terminate-and-stay-resident" (TSR), for Windows as a "virtual device driver" (VxD), for Windows NT as an NT-service, for Linux as a loadable kernel module. Only a memory resident virus may use some "modern" virus techniques like stealth capabilities. For the memory resident virus, one can differentiate between a **fast infector** ([RS1996, p. 174]) and **slow infector** ([CS1995, p. 19], [MR1995]). Both got their name due to the speed they spread. The first one infects every program which is being accessed (read/write) or even all files being listed in a directory listing (e.g. when the "dir" command is being executed). The latter one, in contrast, infects only a file, when it's being written (e.g. during compilation of a new program or some older programs stored their configuration settings directly into the executable file). Therefore, a slow infector may bypass file integrity checkers.

### 1.1.2.3 Special Virus Features

The following **special virus features** ([MR1995]) will be explained briefly:

- stealth technique,
- retro capabilities,
- polymorphism.

Some special virus features can only be used by memory-resident viruses.

**A stealth virus** ([RS1996, p. 173], [MR1995], [CS1995, p. 18]) tries to hide itself by hooking several interrupts like BIOS Int 13h or DOS Int 21h. Assumed, an anti-virus program reads the MBR via BIOS Int 13h to scan for viruses, the virus can intercept this and "redirect" the read call to the saved copy of the original, uninfected MBR. Therefore, the anti-virus program won't find any virus. Or, if a virus scanner scans a file, this file must be opened first. The open call, "redefined" by the virus, will first remove the virus from the file and then call the original open call. After the scanning of the file has been finished, the file will be closed by the virus scanner. And the modified close call will infect the file again.

A **retro virus** ([CN1999, p. 12]) avoids to infect particular file names, like "scan.exe" or "f-prot.exe" (as in most cases those files belong to McAfee VirusScan or F-Prot AntiVirus respectively) as most anti-virus software checks their integrity upon start. This mechanism can be used by non-memory resident viruses, too, of course. A resident virus may even intercept the execution of "scan.exe" and display a "faked" error message like "not enough memory".

A **polymorphic virus** ([CS1995, pp. 20], [MR1995], [RS1996, pp. 174], [CN1996]) is being "encrypted" and changes infection its shape and structure of the (de|en)ryption routine by each infection but the basic functionality is always the same. Here's a very easy example to simply get the basic idea how it works: a CPU has a set of registers, e.g. the accumulator register AX. For example, this register should be set to zero. This can be done by setting the register to zero, i.e. MOV AX, 0. Or by subtracting the current value of the AX register with itself, i.e. SUB AX, AX. Or by the exclusive-or operation, i.e. XOR AX, AX. In short, the effect is just the same, but each operation will result in a different opcode. This technique is also known as "mutation".

### 1.1.3 Macro & Script Viruses

#### 1.1.3.1 Macro viruses

Probably the best definition of a macro virus has been given by Vesselin Bontchev ([VB1997, p. 178]): "A macro virus is a set of one or more macros which set is capable of replication itself recursively".

It's believed, the first macro virus, written by an US-security specialist, was "WM/DMV" and "XM/DMV" - the so called "demo macro virus" for Microsoft Word (WM = Word Macro) and Excel (XM = Excel Macro), back in 1994. The first big "impact" was caused by WM/Concept (1995), because it was the first macro virus found "in-the-wild" ([MH1998, p. 289], [IM1999a, p. 13]), i.e. it was reported by end users - even Microsoft shipped it on CD to customers ([VB1996, p. 97]). We will focus on Word macro viruses here. For doing automatic "tasks", Microsoft invented a macro language called WordBasic (Word 2.0-Word97), and later Visual Basic for Applications. Those macros are stored within the "document" itself, and not as e.g. by Ami Pro<sup>2</sup> in a separate file. So, if you get a document, you'll receive the macros in it, too. Strictly spoken, only template files (.DOT files) can contain macros (Word 2-Word 6). Actually, the virus has to convert a normal word document (.DOC file) first into a template, infect it, and rename the file from .DOT to .DOC, so the user thinks it's just a normal word document file. The DOC → DOT conversion is actually done by Word itself, the macro virus must only set a specific flag. When the user opens such an infected "document", the virus gets activated (mostly) automatically due to several so-called Auto macros (e.g. AutoOpen, AutoExec and AutoClose). The next step is to infect the global template file, in most

---

<sup>2</sup>a text processing software

cases this is the NORMAL.DOT file. As NORMAL.DOT is launched every time Word is started, the macro virus is active every time and is able to infect every word document. But the mentioned Auto macros are not the only way a macro virus can rely on. Other possibilities are shortcuts (like ALT+S), forms or buttons. Many variants of macro viruses have been created automatically: when a word file is being saved, functions from the OLE32.DLL (OLE = Object Linking and Embedding, DLL = Dynamic Link Library) are being used. But as some versions of this DLL are buggy, a file may get slightly damaged when it is saved. Therefore, the macro code might be changed slightly, but the macro virus itself may remain intact and is able to spread. As the macro code has changed, a new variant has been created ([VB1997, pp. 177], [SK1999]).

By Office97, Microsoft introduced a new (macro) language: Visual Basic for Applications ([DAJ1997], [SK1999]). Now, even regular documents can contain macros, so conversion from template to document is no longer necessary. Moreover, Word97 provides an upconversion feature for documents created with older Word releases ([DAJ1997], [VB1997, pp. 188], [VB1998]). So, in most cases macros are converted automatically, too. The commands are not really converted from WordBasic to VBA, but simply speaking just "WordBasic" is placed in front of every instruction. With Service Release 1 (SR1) for Office97, Microsoft tried to make it harder for macro viruses to spread: macros can no longer be copied from the global template into the word document ([VB1998, pp. 157], [JK1998, pp. 144]). But this very weak kind of protection can be bypassed by some tricks, e.g. export the VBA code into a file and re-import this file later into the word document ([KT1999, p. 302]). Upconversion is also done in Office 2000 ([RJZ1999, pp. 223]).

Macro viruses can "snatch" ([VB1996, p. 115], [VB1997, pp. 180]) existing macros, i.e. a macro is replaced by a macro with the same name, which is already present in the global template (e.g. user macros, macros from another macro virus, which has infected the template already before). As an example, many macro viruses have snatched macros from the mostly useless Macro protection tool ScanProt from Microsoft ([VB1997, p. 183]). By macro snatching, a macro virus can mutate to a new variant or a new macro virus. Multiple infections of the normal template can create problems for anti-virus tools, e.g. cleaning it may create a new macro virus if the anti-virus tool thinks it is only infected by one macro virus (see [VB1997, pp. 180] for details). Multiple infections by so called Word Class Infectors<sup>3</sup> are sometimes called "sandwiches" ([KT1999, p. 304]).

Some "advanced features", already known by (DOS) viruses, have been "re-invented" by macro viruses, too. A **stealth macro virus** can display the error message "not enough memory" or a faked dialog box, when the user calls the menu item "Tools/Macro" (this could be used to see which macros exist). The start of the VBA Editor could also be blocked or the for- and background colour is set to white, so that no lines of code can be viewed ([VB1996, pp. 115], [JK1999], [IM1999b, pp. 19]). A **polymorphic macro virus** can change the

---

<sup>3</sup>"Class infectors are Office97 macro viruses that consist of a single module - the class module, which is always named the same (usually ThisDocument)." ([KT1999, p. 301])

code by every infection, i.e. adding a random comment to the code, change some instructions and so on. The polymorphic capabilities are limited, because Word Basic/VBA is rather slow ([VB1996, pp. 112], [VB1997, pp. 192], [ANMK1999, pp. 14], [VBKT2002]). A macro virus is **anti heuristic** if the virus tries to make detection more difficult, e.g. by hiding the macro code in (document) variables, autotexts or encrypted strings ([VB1996, p. 110]).

### 1.1.3.2 Scripting Viruses

Viruses written in VBScript (Visual Basic Script) ([VBM1998, pp. 13]) / Jscript (Windows) or in a shell script language (like bash in Linux/Unix) are called **script(ing) viruses**. "Microsoft VBScript (VBS) is a subset of the Microsoft Visual Basic programming language [...] VBS files can be embedded into HTML documents to make them more interactive. [...]" ([MVO1999, pp. 227]). VBS is supported in MS Outlook, Microsoft Internet Explorer or via the Windows Scripting Host. It's even possible to write a virus, which works with VBA and VBS and can infect Office files and vice versa ([KT1999, pp. 311]). Some other applications, like Corel Draw, offer their own scripting language, e.g. CorelScript, which are used by viruses, too ([NF1999, pp. 7]). Script viruses for Linux/Unix can be written e.g. in a shell script language (like bash) or Perl, Python and so on ([SB1996, p. xxviii]). Some advanced techniques, known from DOS or macro viruses, are possible, too, e.g. polymorphism ([VBKT2002]). Figure 1.3 shows a screenshot of a VBS virus construction kit (construction kits are available for other types of viruses, too).

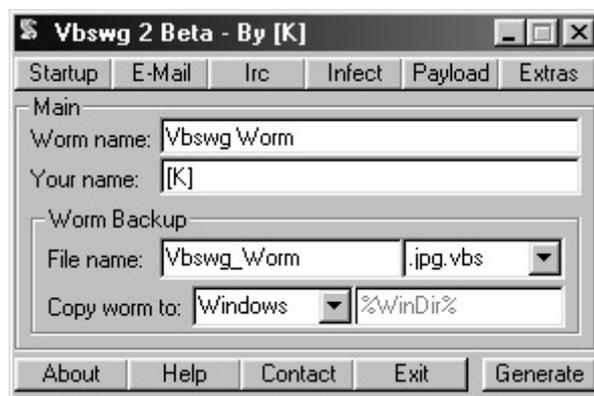


Figure 1.3: Screenshot of VBS virus/worm construction kit, taken from [FSC2003]

### 1.1.4 Worms

The basic difference between a computer virus and a worm is: a worm does not need a host ([RS1996, p. 4]). "The **computer Worm** is a program that is designed to copy itself from one computer to another, leveraging some network medium: email, TCP/IP, etc." ([CN1999, p. 1]). Worms may use

bugs, i.e. buffer overflows ([ECPZ2002]), in some server software to "infect" other machines via the Internet, e.g. MS SQL Server, MS Internet Information Server ([ECPZ2002, pp. 86]) or in FTP daemons or the line printer daemon on Linux/Unix systems ([JK2001, pp. 152]).

According to [CN1999, p. 2-3] Computer Worms can be classified based on the transport and launch mechanism. The transport mechanism are eMail, i.e. using MS Outlook to send itself as an email attachment or the worm may have an SMTP module implemented to create and send mails on its own. A worm may use arbitrary protocols like IRC (Internet Relay Chat), TCP/IP or peer-to-peer networks. If a worm does not "require user interaction in order to gain control of a system" ([CN1999, p. 3]), it's called "self-launching worm". An "user-launched" worm must be started by an user, e.g. double-clicking on an infected email attachment. If a worm uses both mechanism, it's called "hybrid-launch worm". Another classification approach was given by [TM2002, p. 237-242]: he classifies worms by compiler and by method of replication. Most worms are compiled with Visual Basic, C and Delphi. Method of replication includes SMTP (worm creates mail on its own), MAPI (Message API), Outlook or network (e.g. via network shares).

Of course, a worm may implement advanced features, as discussed above, like retro capabilities or polymorphism<sup>4</sup>.

## 1.2 Anti-Virus Technologies

### 1.2.1 Scanner

Virus scanners are the far most used method to detect (and clean) a virus. It may either work on-demand (i.e. the user has to start the virus scanner) or on-access, which means the program runs in the background and scans a file while it's being accessed.

Modern virus scanners must be able to parse various file formats, e.g. different types of executables (i.e. DOS EXE-File, Windows NE/PE- Files), Office files or MIME-encoded files. And of course, self-extractor formats like PkExe or the nowadays very common UPX and archive formats like RAR or ZIP ([KC2002, page 3] [AM2002]). Therefore, an exact file type recognition is needed. Some file types, e.g. images are not susceptible of viruses, so they do not need to be scanned. Or depending on the file type, only certain areas are scanned first; which is faster than doing always a dumb, full scan from the beginning to the end of a file ([IM2000, p. 150]).

Several virus detection methods are possible, and which are used may depend of the type of the virus and/or file type (see above):

**Pattern matching:** for each known virus, a particular sequence of code is "extracted", mostly called pattern, signature or search string, and stored in a virus-definition file (some kind of database). Therefore, the virus scanner "is looking for an exact match which will identify the code as

---

<sup>4</sup>Polymorphism refers to the generated mails here also, e.g. random subject lines.

a virus” ([KC2002, page 5]). To detect variants or minor modifications of a virus, a search string may contain wildcards. Not only the search string is stored, but also information which file types can be infected by this particular virus and at which byte position/offset the search string may occur. This is used to speed up the virus scan process ([IM2000, p. 150]) and to avoid false positives<sup>5</sup>. Moreover, the virus definition file may not only contain the virus signatures, but also some machine code or some pseudo-code for performing various scanning tasks ([IM2000, p. 146]). For identifying a virus, more than one signature could be used; once again, to reduce the likeliness of a false positive ([FF2001, p. 408]). An exact identification is also important for cleaning a virus, otherwise it may happen that the cleaning process removes not only the virus part(s). The basic advantage of pattern matching is that the virus can be named (e.g. ”file infected with XYZ.A virus”), whereas heuristics (see below) may only report ”file looks suspicious” back to the user. The basic disadvantage obviously is that only known viruses can be detected, i.e. the signature has been added to the virus definition file ([FF2001, p. 409]).

**Heuristics:** heuristics is used to ”detect” new viruses. Simply speaking, by the heuristic approach a program is ”analysed” for instructions (or set of instructions) which are known as typical for viruses. Each of such suspicious instruction is given a special weight, which is summed up. If the sum exceeds a particular threshold, the file is regarded as suspect of infection. Another approach is a rule-based system, which ”simply compares found functionality with a set of rules. If a predefined rule is found within the code, the rule-based system returns with a positive result. Depending on the exactness of the complete system, results like generic virus or e.g. VBS/Loveletter variant are realizable” ([MS2002]). ”There are two different ways of applying heuristic rules: static and dynamic. The static method checks the presence of suspicious code fragments (whether they are executed or not). The dynamic method emulates the program and checks which actions are really performed (that is simulation of a virus execution in a virtual environment, frequently called a sandbox or an emulator buffer)” ([IM2000, p. 146], [FF2001, p. 416-417]). Those methods are sometimes also referred as ”passive” and ”active” approach ([RS2002, p. 109]). Both can be combined, too ([FF2001, p. 417]). See also code emulation, below.

**Code emulation:** code emulation was originally developed to detect polymorphic viruses ([FF2001, p. 419], [KC2002, p. 6]). So, if a program is being scanned by the anti-virus program, this program is being executed in a virtual environment (aka ”sandbox”). Therefore, ”when a scanner loads a file infected by a polymorphic virus into this virtual computer, the virus decryption routine executes and decrypts the encrypted virus body. This exposes the virus body to the scanner, which can then search for signa-

---

<sup>5</sup>false positive = a file is reported as infected, although it is clean. False negative = a file is reported as virus free, although it is not

tures in the virus body that precisely identify the virus strain” ([CN1996, p. 5]). As mentioned above, code emulation is also used together with heuristics. Of course, code emulation is slow ([FF2001, p. 420]). So, code emulation should only be used when really needed ([IM2000, p. 150-151]).

### 1.2.2 Integrity Checker

An **integrity checker** basically generates a checksum for files, sectors (i.e boot sector) and the macros, stored in e.g. an Office document (it would make no sense to create a CRC for the whole Office file, obviously). The checksums are being stored in a kind of database and later being compared. If a checksum does not match, a file has been modified (which could be caused due to a virus infection) . Obviously, when generating the checksum, it must be assured the file is clean ([FF2001, p. 411-412], [AN1999]).

### 1.2.3 Behaviour Blocker

A **behaviour blocker** runs in the background and monitors the execution of the currently running program(s) on the computer. If a program tries to do a suspicious action (e.g. open a file and appending code or formatting hard disc), this will be intercepted. The behaviour blocker may then terminated this program or ask the user which action should be taken (e.g. allow, do not allow, move program into quarantine) ([CN2002], [FF2001, p. 410-411]). But for most users this decision is a ”tough choice” and behaviour blocking may generate a high level of false positives ([FF2001, p. 411]), although some techniques are possible to reduce the likelihood of false positives ([LL2001]).

## 1.3 Anti-Virus Strategy

Nowadays, a basic anti-virus strategy is a 3-tier approach:

- tier 1: the desktop,
- tier 2: file & print, email or web server(s),
- tier 3: the internet gateway, like mail gateways or web proxy servers.

A virus should be stopped as early as possible, before it can enter the network (i.e. tier 3). According to [ICSA2002, p. 24-25] since 2000 more than 80% of the virus incidents have been caused by infected email attachments, whereas diskettes as source of infection are next to nothing. Of course, encrypted emails / attachments can not be checked at this level (this has to be done by an on-access scanner running on the desktop). Virus scanning requires lots of resources, so this task should probably be off-loaded onto another machine. Anti-virus software on the gateway must take precautions to not suffer from a Denial-of-Service (DoS) attack by special crafted mails and/or mail attachments (e.g. the (in)famous 42.zip attack<sup>6</sup>).

<sup>6</sup>see <http://www.corpit.ru/pipermail/avcheck/2001q3/000110.html> for a discussion on the avcheck and amavis-user mailing list

As files, esp. documents are shared via file server(s), those are a vector for distributing infected documents (some viruses/worms use network shares to propagate itself). Therefore, on-access scanning of file servers is the next line of defense (tier 2).

The last resort is the desktop, e.g. for scanning an encrypted file when it's being decrypted.

A full anti-virus strategy is beyond the scope of this paper. Of course, a good backup/restore concept and user education plays an important role in such a concept. Only with a backup, erased data (caused by accident, hardware failure or the payload of a virus) can be restored. User education should help to minimize the number of people who double-click on any email attachment. To reduce the risk of virus infection (and propagation), you may choose "safe" file formats (e.g. plain ascii text or PDF<sup>7</sup> instead of Word), "safe" applications (e.g. PegasusMail instead of Outlook or OpenOffice instead of MS Office) or Linux/FreeBSD instead of Windows. But, of course, changing software/OS may not be an easy task.

---

<sup>7</sup>PostScript/PDF viruses are possible and some "proof-of-concept" viruses exist. So, the PS/PDF reader must not execute malicious instructions.

## Chapter 2

# Server-based Virus Protection on Unix/Linux

This chapter outlines the requirements, possible means to integrate 3rd party virus scanners and whether those requirements are full-filled or not.

### 2.1 Requirements & Aims

As the title already implies, the thesis is focused on server-based anti-virus solutions running on Unix/Linux servers, i.e. protection for Internet gateways (tier 3) and file servers (tier 2) to mainly protect Windows clients within the internal network. So, we will discuss Open Source solutions for Linux/Unix servers acting as

- Mail servers, running sendmail or postfix (chapter 4, p. 49). This topic will be covered shortly, as many solutions exist (e.g. AMaViS, qmail-scanner, exiscan, MailScanner) and some of them for many years (the AMaViS project was started back in 1997).
- File and print servers, running Samba (chapter 5, p. 55). Not many solutions are available until now; some features and implementation details of samba-vscan will be discussed, which up-to-now is the only Open Source solution supporting several anti-virus products.
- Proxy server running Squid (chapter 6, p. 71) . Again, OSS solutions are rare. Three concepts by example will be presented shortly.

As those OSS solutions are not anti-virus products per se, but acting as "clue code" between the service and one (or more) virus scanners, how the integration can be done will be discussed first. The following requirements should be fulfilled:

- easy integration, e.g. into shell or Perl scripts
- open, non-proprietary protocol, to be independent from anti-virus vendor(s)

- low implementation and maintenance efforts, to reduce costs
- load balancing / load separation, to not suffer from resources loss which slows down the main service (e.g. mail server)
- high performance, to reduce latency caused by the virus scan process in whole

## 2.2 Integration of Anti-Virus Products

Several means to integrate an anti-virus product into any 3rd party application exist. Each has it's pros and cons, some can be implemented very easily, others are more time consuming.

### 2.2.1 Command line Scanner

A command line scanner can be used on demand to scan a specific directory or the whole disk. Or via a cron job, calling a specific script, for a scheduled scan, e.g. every day at 12pm.

Depending on the return value (or exit status) of the called program it's possible to determine whether an infection was found or not. As an example, the list of return codes (shortened) for H+BEDV AntiVir/Linux (as of version 2.07):

- 0: Normal program termination, nothing found, no error
- 1: Found infected file or boot sector
- 2: A signature was found in memory

Calling a virus scanner is possible from any (shell) script, e.g.

```
#!/bin/sh
/usr/sbin/antvir /path/to/check
ret=$?
if [ $ret -eq 0 ] ; then
    echo "No virus found"
elif [ $ret -eq 1 ] ; then
    echo "Virus found"
else
    echo "An error occurred"
```

Checking only for the return code has one major drawback - you can't get the virus name(s). Of course, grepping the output is possible, like in the Perl example below, taken from the hbedv module<sup>1</sup> of the AMaViS project<sup>2</sup> (simplified).

```
[..]
    chop($output = '$antivir -allfiles -noboot -s -z $TEMPDIR/parts');
```

<sup>1</sup>full source at <http://cvswb.amavis.org/amavis/amavis/av/hbedv>

<sup>2</sup>A Mail Virus Scanner, <http://www.amavis.org/> - Perl script for virus scanning at the email gateway level.

```

    $errval = retcode($?);
    do_log(2,$output);
    if ($errval == 0) { # no errors, no viruses found
        $scanner_errors = 0;
    } elseif ($errval == 1) { # no errors, viruses discovered
        $scanner_errors = 0;
        if ( $output =~ /ALERT:/ ) {
            @virusname = ($output =~ /ALERT: \[(\S+)\s.*?\]/g);
        }
    }
}
[.]

```

The basic format is "ALERT: [<name> <type>] some text", whereas <name> is the name of the virus and <type> e.g. virus or dialer. But those types are subject to change<sup>3</sup>.

So, are the requirements fulfilled?

- ✓ easy integration (partly), at least if used in scripts. Calling a command line scanner using exec(2) family in a C program may not always be possible (e.g. in the samba-vscan case).
- ✓ the implementation and maintenance efforts are in general low. Only information about the command line switches, the return values and the output being matched is needed. From the authors experience as the unofficial AMaViS av-subsystem maintainer: once in a while, it may happen anti-virus vendors change return values or the output without prior notice.
- ✗ no open "protocol", each anti-virus product has it's own set of return values and output. Switching vendor means changing existing scripts.
- ✗ load balancing / separation is not possible. The virus scanning task has to be done on the same machine as the service is running and could not offloaded to another machine.
- ✗ performance is low. The program start needs time, i.e. creating the program executing environment, self-check, loading the anti-virus database.

### 2.2.2 Application Programming Interface

An application programming interface (API) allows a 3rd party developer to integrate a virus scanning facility in his program(s). Many anti-virus products offer such an API, like the one(s) from Sophos, Trend Micro, H+BEDV or Network Associates. But for most (nearly all) products, details about the API is not freely available, i.e. only after signing a Non-Disclosure-Agreement (NDA). To my knowledge, the only vendor which provides a complete documentation of its API is Sophos Plc, which is called SAVI (Sophos AntiVirus Interface)

<sup>3</sup>based on an email from John Ogness of H+BEDV Datentechnik GmbH, Germany.

([SAVI, page 13]): "SAVI consists of a set of interfaces and enumerators that provide access to various objects which are used internally by SAVI. [..]"

The interfaces are retrieved:

- By querying the class factory, *or*
- By allowing COM to supply them automatically (only when using C++ syntax), *or*
- From the SAVI interface itself.

The interfaces can be used with C++ syntax or C syntax."

SAVI provides various functions, e.g. *Initialise* for initialising an SAVI object, *SweepFile* to scan a single file for viruses or *DesinfectFile* for attempt to disinfect a file ([SAVI, page 63]). For illustration, a sample code on how to initialise SAVI using the C programming language ([SAVI, page 19-20]):

```

CISavi2* pSAVI;
CISweepClassFactory2*
pFactory; HRESULT hr;
const char* ClientName = "SAVIDemo";

/*
 * Load the SAVI DLL and then request a class factory
 * interface.
 */
hr=DllGetClassObject((REFIID)&SOPHOS_CLSID_SAVI2,
                    (REFIID)&SOPHOS_IID_CLASSFACTORY2,
                    (void **)&pFactory );
if( hr==SOPHOS_S_OK )
{
    /*
     * Ask the class factory for a CSAVI2 interface.
     */
    hr=pFactory->pVtbl->CreateInstance(pFactory, NULL, &SOPHOS_IID_SAVI2,
                                    (void**)&pSAVI );

    /*
     * Drop the factory immediately, we don t need it
     * again in this example.
     */
    pFactory->pVtbl->Release(pFactory);
    /*
     * Did we get the CSAVI2 interface we requested?
     */
    if( hr==SOPHOS_S_OK )
    {
        /*
         * Ask SAVI to initialise itself.

```

```

    */
    hr=pSAVI->pVtbl->InitialiseWithMoniker(pSAVI, ClientName);
    /*
    * If the initialisation failed, then release the
    * SAVI interface and
    * set the pointer to NULL.
    */
    if( SOPHOS_FAILED(hr) )
    {
        printf("ERROR: Initialise [%ld].", (long)hr);
        pSAVI->pVtbl->Release(pSAVI);
        pSAVI = NULL;
    }
}
}

```

Now, let's have a look whether the requirements are met or not.

- ✓ good performance, as only the library has to be imported once and can then be used. Loading library is faster than starting a program.
- ✗ easy integration is not possible, e.g. it can not be used in scripts
- ✗ each anti-virus vendor has its proprietary API. So, changing the anti-virus vendor means rewrite from scratch.
- ✗ high implementation efforts. Needed to get in touch with API, efforts depend on the complexity of the API and on API documentation. If API is stable, changes are rather unlikely so maintenance efforts should be rather small, assumed the API has not been completely redesigned.
- ✗ load balancing / splitting not possible, as the library can only be called if it's installed on the same host on which the service being protected is run.

### 2.2.3 Client-server Communication

Using a command line scanner has once again a major drawback: speed. Even if the OS/filesystem provides a good caching mechanism, the start takes some time. This can be avoided when the program is run as a daemon. "Daemons are processes that live for a long time. They are often started when the system is bootstrapped and terminate only when the system is shutdown. We say they run in the background, because they don't have a controlling terminal. Unix systems have numerous daemons that perform day-to-day activities" ([WRS1992, page 415]). So, the anti-virus daemons have to load the virus signature database only once on start-up, and not for each scan.

As an example, using Sophie (a daemon, using SAVI) with AMaViS the speed-up is about 2.5x compared to the command line scanner Sophos Sweep:

”Based on the average delay times (as logged by postfix for the vscan transport<sup>4</sup>), I am experiencing a roughly 2.5x speed-up in mail processing compared to Sophos sweep. This is on a real life, production mail server, not some fancy benchmark, and it’s the first time ever I have seen log entries with ’relay=vscan, delay=0’ :-)” [LH2001].

We refer to the daemon, as it provides a virus scanning service, as a server, which waits for requests from (any) program. Such a program is called client, as it sends a request to the server to scan a file or directory.

Client and server communicate via (BSD) sockets, either Unix Domain sockets or TCP sockets. By using Unix Domain sockets, communication is limited to the host, i.e. client and server have to run on the same host.

The used protocol (for communication) could be either a ”proprietary” or standardised one (e.g. the Content Vectoring Protocol or Internet Content Adaption Protocol), which will be discussed in the following to sections.

### 2.2.3.1 Proprietary Protocols

Currently, most anti-virus products running as daemon use a ”proprietary” protocol. `samba-vscan` (chapter 5, page 55), a program for on-access virus scanning with the Samba file server, supports seven anti-virus products, which means five different communication protocols (one pair use the same protocol). On the one hand, this means some work for each virus scanner, but on the other hand most of those protocols are easy to implement.

The protocol for the OpenAntiVirus Scanner daemon is very simply and straightforward. It waits on port 8127 for a connection and expects e.g. the ”SCAN *filename|path*” command<sup>5</sup>. The response may either ”OK” (file is clean), ”FOUND *virus-name*” or ”ERROR: *error message*”. The connection is then closed by the server. For illustration, a simple telnet session to scan the file `eicar.com`, which contains the EICAR<sup>6</sup> Test File<sup>7</sup> virus (not a real virus).

```
$ telnet localhost 8127
Connected to localhost.
Escape character is '^'.
SCAN /tmp/eicar.com
FOUND: Eicar-Test-Signature
Connection closed by foreign host.
```

Some products, ”re-use” existing protocols for their own purpose, e.g. F-Prot Daemon uses the Hyper Text Transfer Protocol (HTTP) 1.0, as specified in RFC1945 [RFC1945]. The F-Prot Daemon binds on port 10200 (up to 10204). F-Prot Daemon supports only the GET method to send the name of the file to be scanned. The request is therefore more simple than the one specified in RFC1945 [RFC1945, chapter 5]:

<sup>4</sup>the vscan transport is the one calling AMaViS

<sup>5</sup>other methods are POST and FILTER

<sup>6</sup>European Institute for Computer Antivirus Research, <http://www.eicar.org>

<sup>7</sup>available at [http://www.eicar.org/anti\\_virus\\_test\\_file.htm](http://www.eicar.org/anti_virus_test_file.htm)

```
Request = Request-Line
        CLRF
```

The Request-Line is simplified, too:

```
Request-Line = Method SP Request-URI SP HTTP-Version CLRF
```

```
Method = GET
```

```
Request-URI = abs_path
```

whereas `abs_path` is an absolute file name here. RFC1945 [RFC1945, section 5.1.2] mentions: "The Request-URI is transmitted as an encoded string, where some characters may be escaped using the "% HEX HEX" encoding defined by RFC 1738. The origin server must decode the Request-URI in order to properly interpret the request." Of course, this applies here, too.

The response send back by the daemon complies to RFC1945 [RFC1945, section 4.1] "Full Response" definition. The "entity body" is XML<sup>8</sup> output, as demonstrated by the following simple telnet session:

```
$ telnet localhost 10200
Connected to localhost.
Escape character is '^]'.
GET /tmp/eicar.com HTTP/1.0

HTTP/1.0 200 Ok
Server: fprotd
Date: Fri, 10 Jan 2003 14:00:52 GMT
Content-Type: text/plain
Connection: close

<?xml version="1.0" encoding="ISO-8859-1">
  <!DOCTYPE fprot-results PUBLIC "" "">
  <fprot-results version="0.0" engine="3.11b">
    <arguments>
      <arg></arg>
    </arguments>
    <filename>/tmp/eicar.com</filename>
    <detected type="malware">
      <name>EICAR_Test_File</name>
      <accuracy>8</accuracy>
      <disinfectable>yes</disinfectable>
    </detected>
    <summary code="11">infected</summary>
  </fprot-results>
Connection closed by foreign host.
```

Both protocols as shown as example here have at least one drawback: they accept a file name only (ScannerDaemon accepts pathname, too). So, they must

<sup>8</sup>Extensible Markup Language, see <http://www.w3.org/XML/>

run on the same host as the client. Therefore, load separation is not possible (i.e. running an Mail Transfer Agent on host X and the virus scanning facility on host Y).

Once again, are the requirements fulfilled?

- ✓ easy integration, mostly yes, as most protocols are simple and may even used within scripts (e.g. with tools like netcat/nc).
- ✓ the implementation and maintenance efforts are relatively low, as the protocols are simple or existing code can be re-used (e.g. XML libs for parsing XML output).
- ✓ good performance, as forking a child is faster than creating a complete program context. Self-check and loading virus database must be done upon startup of the daemon only.
- ✗ although the protocol is open (e.g. derived from HTTP), they are still proprietary, as each anti-virus program uses its own protocol. Switching vendor means re-write of the client
- ✗ load balancing / load separation can not be done with very simple protocols, which only accept the file name (and not the file contents).

### 2.2.3.2 Generalized Frameworks

All mentioned techniques so far have two (major) drawbacks.

- Each anti-virus product has it's own set of return codes, or specific API or communication protocol. This means a lot of work to develop and maintain support for each anti-virus product if you are an 3party application developer. So, e.g. the AMaViS program ships with more than 25 antivirus-specific modules<sup>9</sup>.
- In some environments it may be desired to have service separation on a per host basis, i.e. running email server, proxy server and anti-virus server each on an own host. This is only possible, when/if the complete data of the file to be scanned is transfered via network. Of course, the network bandwidth could be the bottle-neck then, but this could be avoided by using more than one virus-scanning host (i.e. per subnet/LAN segment) and load-balancing.

Probably the most well-know protocol, which isn't "flawed" with the issues mentioned above, is the "Content Vectoring Protocol" (CVP<sup>10</sup>) by Checkpoint Software, only for use with their Firewall-1 product. The other one is the Internet Content Adaption Protocol (ICAP), mainly developed by Network Appliances Inc and Akamai Technologies. Of course, both are not limited to

<sup>9</sup>see <http://cvsweb.amavis.org/amavis/amavis/av/>

<sup>10</sup>please do not mix with the "Certificate Validation Protocol", abbreviated as CVP, too. See <http://www.ietf.org/internet-drafts/draft-ietf-pkix-cvp-01.txt>

virus scanning at all; they provide a generalized framework for various kinds of content inspection and modification.

CVP is part of Checkpoint's OPSEC (Open Platform for Security), <http://www.opsec.com/>. As taken from [CVP2002, page 2]: "CVP (Content Vectoring Protocol) inspection is an integral component of VPN-1/FireWall-1's Content Security feature. It enables third-party Content Vectoring Servers to examine all files transferred for various protocols and considerably reduces the vulnerability of protected hosts. CVP configuration (which files to inspect, how to handle the invalid files) is available for all resource definitions. All VPN-1/FireWall-1 auditing tools are available for logging CVP inspection and issuing alerts if necessary." (see figure 2.1)

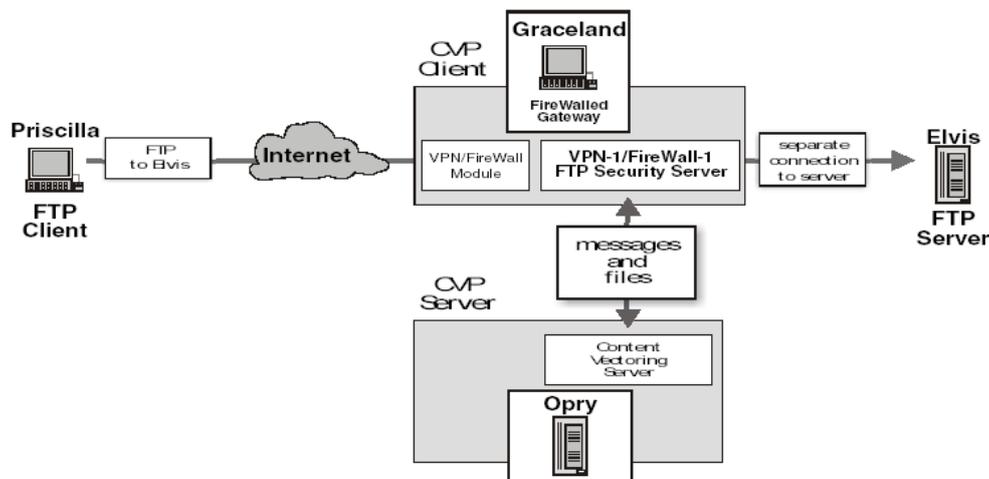


Figure 2.1: Connection invoking a Content Vectoring Server [CVP2002, page 2]

Figure 2.1 shows an FTP client, running on host "Priscilla", which is connected to the FTP server on "Elvis" via the firewalled gateway "Graceland". We assume, the client tries to retrieve a file from the FTP server, therefore the Firewall-1 Security Server invokes the Content Vectoring Server (on "Opry"). So, VPN-1/Firewall-1 sends the file to the CVP server. The latter one performs a virus scan and may optionally send back the auto-cleaned file. Depending on the policy setting, the Security Server allows or disallows the file transfer.

The data flow is illustrated in 2.2 [CVP2002, page 7].

1. Input flow – from the source of the connection to the CVP client
2. Server flow – from the CVP client to the CVP server
3. Client to destination – from the CVP client to the connection destination
4. Server to destination – from the CVP server to the connection destination  
Destination flow is a combination of the client to destination and server to destination flows
5. Source flow – from the CVP server to the source of the connection

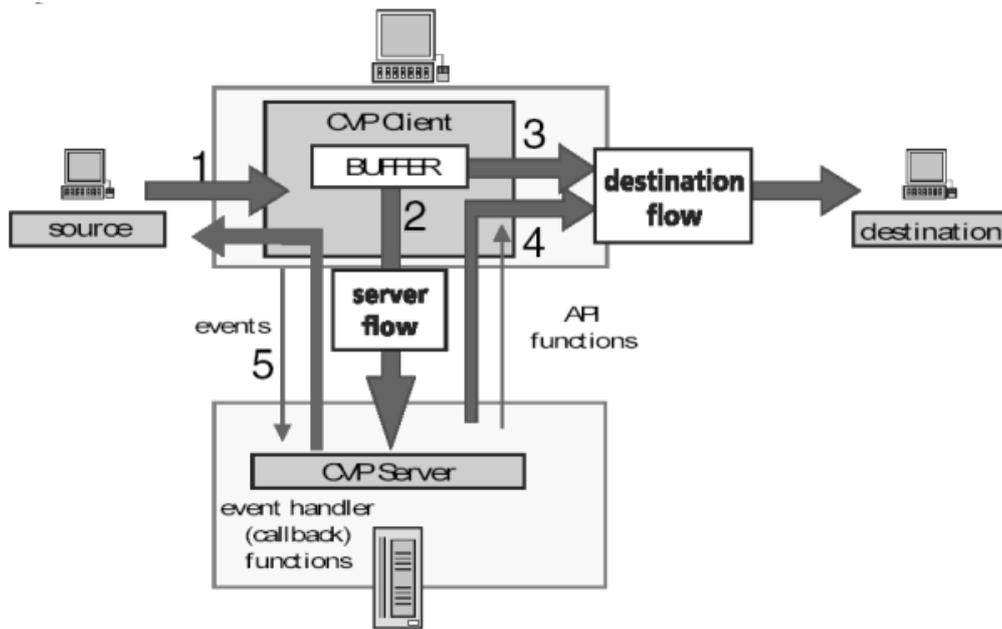


Figure 2.2: CVP data flows [CVP2002, page 6]

Actually, CVP is a somewhat complex protocol (at least when compared with the ICAP protocol, as discussed below). Moreover, the SDK and documentation is written for developing a CVP server and not for a CVP client. As an example the SDK ships with several example servers in C, an example client is only available as a binary. But for our purpose, a client must be developed, not a server. So, the documentation/SDK is not that valuable. It seems, it's Checkpoint's aim to not provide any information on how to write a client [KA2001].

In contrast, the Internet Content Adaption Protocol (ICAP) is freely available, i.e. the technical documentation can be downloaded without prior registration. Moreover, the example server implementation is licensed under the terms of the GNU Public License (GPL)<sup>11</sup>, probably the most prominent and most widely used<sup>12</sup> Open Source/Free Software license. The ICAP protocol is dissected in the next chapter.

<sup>11</sup><http://www.fsf.org/licenses/gpl.html>

<sup>12</sup>see freshmeat.net license breakdown statistic, <http://freshmeat.net/stats/license>. "freshmeat maintains the Web's largest index of Unix and cross-platform software, themes and related 'eye-candy', and Palm OS software", taken from <http://freshmeat.net/about/>

# Chapter 3

## ICAP

In this chapter, the ICAP protocol will be dissected, i.e. who introduced it? What were the requirements? How does it work? At the end of this chapter, the example "icap-client" will be discussed, i.e. its usage and implementation.

### 3.1 Overview

#### 3.1.1 Introduction

ICAP was introduced by the so called ICAP forum back in 1999. The ICAP forum is a coalition of Internet businesses and was co-founded (and still co-chaired) by Network Appliances and Akamai Technologies. The requirements to be full-filled are [ICAP01, page 2]:

- "Be simple.
- Be scalable.
- Use existing infrastructure.
- Be modular in its service. That is, services must be able to be added and subtracted without affecting the intervening architecture or its performance.
- Use existing communication methods and standards.
- Provide resource savings by leveraging edge services."

In short, "ICAP in its most basic form is a 'lightweight' HTTP based remote procedure call protocol. In other words, ICAP allows its clients to pass HTTP based (HTML) messages (Content) to ICAP servers for adaptation. Adaptation refers to performing the particular value added service (content manipulation) for the associated client request/response." [ICAP01, page 2]

The benefits of ICAP are [ICAP01, page 3]:

- "ICAP leverages existing equipment available today. In fact, if NetCache (a proxy appliance) proxies are already installed, then no new equipment is necessary, with the exception of the ICAP servers.

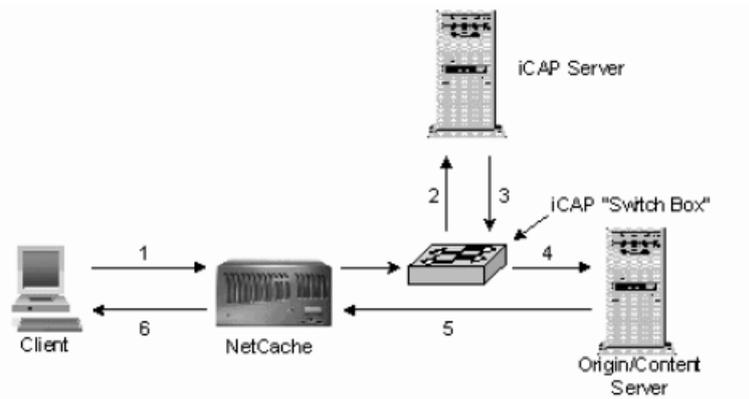


Figure 3.1: Request Modification [ICAP01, page 7]

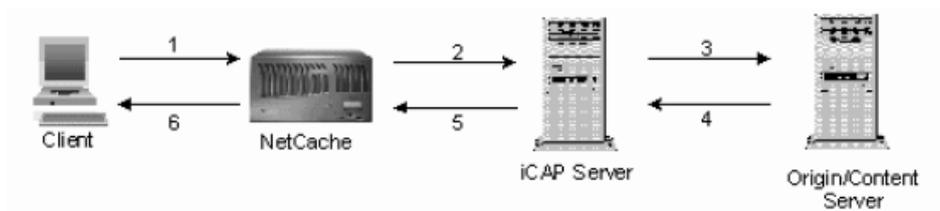


Figure 3.2: Request Satisfaction [ICAP01, page 7]

- ICAP is HTTP based, enabling access through security barriers that only allow port 80 traffic. Therefore, no security changes to the existing network are likely.
- ICAP is an *open protocol* and allows any server or application provider to implement it. ICAP is easy to implement since it leverages Apache code. ISPs and enterprises can then choose the appropriate value-added application provider.
- ICAP can also collect client interest information for use in targeting more focused advertising toward these individuals.
- ICAP off-loads these value-added services to ICAP servers, freeing up the resources of the Web servers. This reduces the access times on these sites.
- ICAP simplifies the implementation, reliability, and scalability of value-added services. ICAP leverages edge device and infrastructure to deliver edge-based value-added services that require content adaptation.”

Services to be implemented by using ICAP are [ICAP01, page 4-6]:

- "Virus scanning". Virus scanning can be performed "on-the-fly". If used for scanning Web traffic, only new traffic will be scanned. Previously scanned content, "marked" as virus free, can be cached by any web cache (like Squid), which improves performance.
- "Markup Language Translation.
- Advertising Insertion
- Human Language Translation.
- Content Filtering
- Data Compression"

### 3.1.2 Architecture

ICAP basically offers four operations: two for modifications of an HTTP request (header), and two for HTTP response (body).

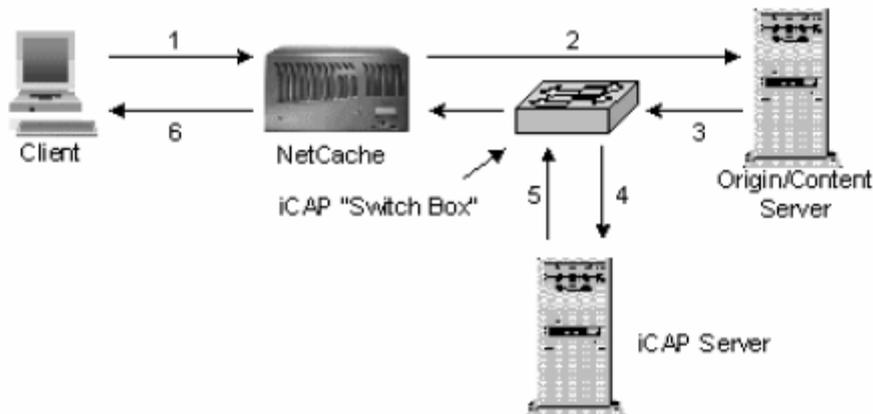


Figure 3.3: Response Modification [ICAP01, page 8]

Via the **Request Modification** method the request issued by a client may be modified by an ICAP server before the request will eventually fulfilled by the origin server (see figure 3.1). Let's assume, the client wants to visit the banned-host.com web site, so the browser sends an HTTP GET request, which will be redirected to ICAP server by the ICAP "switch box". As the site "banned-host.com" is in the list of banned URLs, the GET request will be rewritten to retrieve an error message (stored on the proxy server). As another example, the ICAP server could "filter out" sensitive data from the HTTP (GET) request before it will be sent back to the proxy, which will then send this request to "the outside world". **Request Satisfaction** (see figure 3.2) works quite similar, but the (probably) modified request is sent directly to the origin server by the ICAP

server (and not send back to the proxy server) and also the response by the origin server will be sent back via the ICAP server.

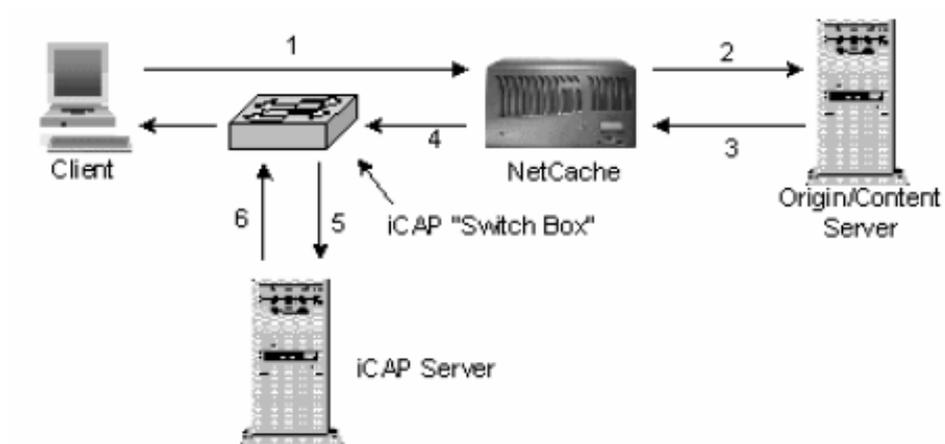


Figure 3.4: Result Modification [ICAP01, page 8]

The **Response Modification** (see figure 3.3) and **Result Modification** (see figure 3.4) are very similar. Here, the request of a client is answered by the origin server and the response is then directed to the ICAP server for modification (if any).

Service	Request Modification	Request Satisfaction	Response Modification	Result Modification
Content Filtering	Yes	Yes	Yes	Yes
Gateway Translation	Yes		Yes	
Language Translation	Yes	Yes	Yes	
Virus Scanning			Yes	
Ad Insertion	Yes	Yes	Yes	Yes
Data Compression			Yes	Yes

Table 3.1: Service Architecture Summary [ICAP01, page 9]

Product	Windows	Solaris	Linux
Symantec AntiVirus Scan Engine	Yes	Yes	Yes
WebWasher (NAI/CAI Engine)	Yes	Yes	Yes
Finjan SurfinGate for Web	Yes	Yes	No
TrendMicro InterScan WebProtect for ICAP	No	Yes	No

Table 3.2: Available ICAP AntiVirus Servers

### 3.1.3 ICAP-enabled Anti-Virus Solutions

For this thesis, the Symantec AntiVirus Scan Engine and WebWasher CSM (with the virus scan engine from Network Associates/McAfee) have been used. Table 3.2 shows currently available ICAP AntiVirus Servers.

#### 3.1.3.1 Symantec AntiVirus Scan Engine

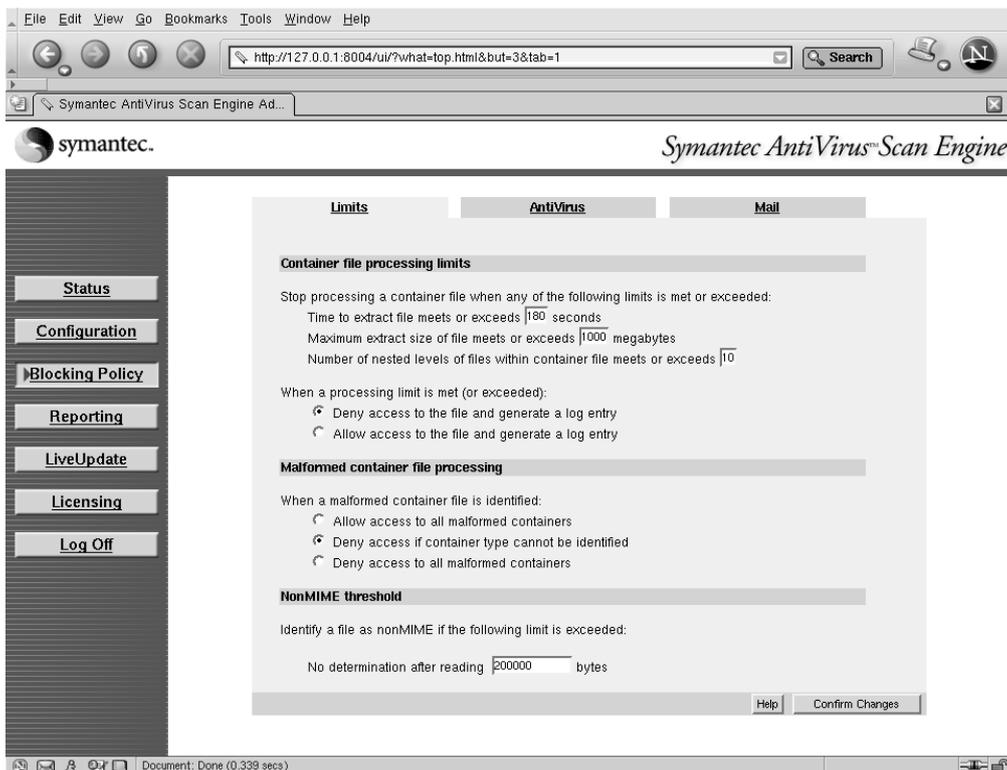


Figure 3.5: Web-frotrontend of Symantec AntiVirus Scan Engine

The Symantec AntiVirus Scan Engine (SAVSE) 4.x<sup>1</sup> was the first ICAP enabled anti-virus solution I received from an anti-virus vendor, which really had

<sup>1</sup>see <http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=173&EID=0>

a working ICAP support<sup>2</sup>. So, most testing and development of the ICAP client has been done using SAVSE. According to [SAVSE02, p. 14], "the Symantec AntiVirus Scan Engine provides virus scanning and repair capabilities to any application on an IP network, regardless of platform, using one of three protocols. Any application can pass files to the Symantec AntiVirus Scan Engine for scanning, which in turn scans the files for viruses and returns a cleaned file if necessary". A scan request can be send via SAVSE's own native protocol, Internet Content Adaption Protocol (ICAP) and Remote Procedure Call (RPC)<sup>3</sup> ([SAVSE02, p. 15]). While it's not possible to mention all features and configuration settings, I'd like to point out the following settings:

- ICAP scan policy allows you to configure the action taken by SAVE, i.e. file is only scanned, scan and delete or scan and repair ([SAVSE02, p. 62])
- In-memory file limits, as SAVSE has it's own in-memory file system used for decomposing and scanning for container and archive files, which is faster than on-disk scanning ([SAVSE02, p. 74]).
- Limits for container files (e.g. archive file), i.e. maximum file size, maximum number of nested archive and maximum amount of time for decomposing ([SAVSE02, p. 87]).

As already mentioned, SAVSE may use the so-called native protocol, which "is a request/reply-based protocol. The protocol version, the request, and the file are all transmitted by the client upon connection with the scan engine. The reply consists of a reply code, scan results, and the file if the file has been modified" ([SSS02, p. 72]). The basic syntax of a client request is:

```
Version 2<CRLF>
<socket-command><CRLF>
<filename><CRLF>
<filesize><CRLF>
<filesize-bytes-of-data>
```

And for the server response:

```
Reply-Code<CRLF>
<scan-results>
<receive-file>
```

### 3.1.3.2 WebWasher Content Security Management (CSM) Suite

The WebWasher (WW) Content Security Management (CSM) Suite provides the most features of the WebWasher product family<sup>4</sup>: Internet Access Management, Internet Content Filtering, E-Mail Filtering and Reporting. Spam

<sup>2</sup>the previous version had broken ICAP support. And I missed the fact, WebWasher offers anti-virus capabilities via third party anti-virus modules.

<sup>3</sup>proprietary implementation

<sup>4</sup>see [http://www.webwasher.com/enterprise/products/webwasher\\_products/index.html?lang=de\\_EN](http://www.webwasher.com/enterprise/products/webwasher_products/index.html?lang=de_EN)

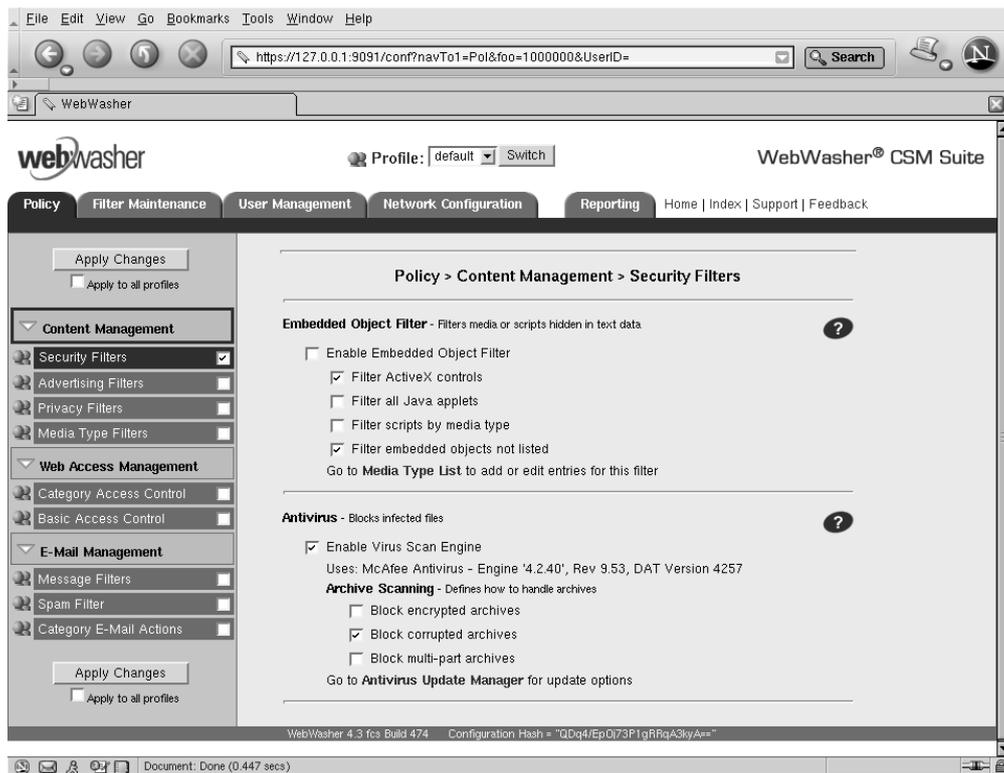


Figure 3.6: Web-frontend of Web Washer CSM Suite

Filtering and Virus Scanning are optional. I used the WW CSM suite with the Network Associates/McAfee virus engine. WW acts itself as ICAP server, but also as ICAP client (e.g. the Internet Content Filtering module can be configured to not use the internal WW-ICAP server but an external one). Moreover, WW CSM can be used with Squid, but provides an own HTTP proxy, too (without caching capabilities).

## 3.2 Technical Specification

### 3.2.1 Overview

The ICAP protocol is specified in RFC3507, published in April 2003 as an informational memo. We will discuss relevant information for performing AV scanning tasks only. For full details please refer to RFC3507 and related RFCs, esp. RFC 2119 (keywords used in RFCs) and RFC 2616 (HTTP/1.1).

Actually, RFC3507 only mentions the OPTIONS (querying features of the ICAP server), REQMOD (Request Modification) and RESPMOD (Response Modification) methods. Request Satisfaction and Result Modification, as named in 3.1.2 (page 25), are not specified. As only the RESPMOD method will be used for virus scanning, we will confine on this method to the essentials. The abstract of [RFC3507] gives the outline of the ICAP protocol:

”ICAP, the Internet Content Adaption Protocol, is a protocol aimed at providing simple object-based content vectoring for HTTP services. ICAP is, in essence, a lightweight protocol for executing a ’remote procedure call’ on HTTP messages. It allows ICAP clients to pass HTTP messages to ICAP servers for some sort of transformation or other processing (’adaptation’). The server executes its transformation service on messages and sends back responses to the client, usually with modified messages. Typically, the adapted messages are either HTTP requests or HTTP responses.”

Although ICAP has several similarities to HTTP (semantics and usage), ICAP is neither HTTP nor is it an application protocol that runs over HTTP [..] The default port is 1344, but others may be used” ([RFC3507]). An ICAP request is an ICAP message sent from the client to the ICAP server, an ICAP response is an ICAP message sent from ICAP server to client. An ICAP message consists of an ICAP header and maybe an ICAP body, which contains an encapsulated HTTP message. In our case, the HTTP message contains the HTTP request and HTTP response. As we use ICAP not only for HTTP purposes, our ICAP client must create ”forged” HTTP request and HTTP response. ICAP messages conform to the generic message format as of RFC 2822 ([RFC3507, section 4.1]).

Details will be explained by an example ICAP request and an example ICAP response<sup>5</sup>.

### 3.2.2 ICAP request

In the following request, the file /tmp/test.txt should be scanned by an ICAP anti-virus server. The file test.txt contains only the line ”TEST TEST TEST” (without the line terminator \n !). Line numbers in front have been added to explain things more easily.

```
01: RESPMOD icap://localhost/avscan ICAP/1.0
02: Host: localhost
03: Encapsulated: req-hdr=0, res-hdr=30, res-body=109
04:
05: GET /tmp/test.txt HTTP/1.1
06:
07: HTTP/1.1 200 OK
08: Content-Type: application/octet-stream
09: Content-Length: 14
10:
11: e
12: TEST TEST TEST
13: 0
14:
```

Line 01 to 14 shows the complete ICAP request message, which consists of the ICAP header (01-03) and the ICAP body (05-14). Line 04 is empty (i.e., a

<sup>5</sup>these are ”real-life” examples, created using the sample ICAP client, which will be discussed in 3.3 on page 35

line with nothing preceding the CRLF), indicating the end of the ICAP header.

The header starts with a request line (01), which contains the method (here RESPMOD), the URI of then ICAP resource, and an ICAP version string ([RFC3507, section 4.3.2]). The ICAP URI is specified as

```
ICAP_URI = Scheme ":" Net_Path [ "?" Query ]
Scheme = "icap"
Net_Path = "://" Authority [ Abs_Path ]
Authority = [ userinfo "@" ] host [ ":" port ]
```

The request line is followed by two more headers (02-03). The "Host" header is required ([RFC3507, 4.3.2]). The "Encapsulated" header is a must for every ICAP message ([RFC3507, 4.4.1]), and will be explained later.

The ICAP body contains encapsulated HTTP message: "The ICAP encapsulation model is a lightweight means of packaging any number of HTTP message sections into an encapsulating ICAP message- body, in order to allow the vectoring of requests, responses, and request/response pairs to an ICAP server. [...] Encapsulated bodies MUST be transferred using the 'chunked' transfer-coding described in Section 3.6.1 of [4]" ([RFC3507, 4.4]). Before we discuss the ICAP body, here the syntax for chunked transfer encoding ([RFC2616, 3.6.1])<sup>6</sup>:

```
Chunked-Body = *chunk
              last-chunk
              trailer
              CRLF

chunk        = chunk-size CRLF
              chunk-data CRLF

chunk-size   = 1*HEX

last-chunk   = 1*("0") CRLF

chunk-data   = chunk-size(OCTET)
trailer      = *(entity-header CRLF)
```

So, the ICAP body consists of the following:

- HTTP request header (line 05). Normally, that's the request from a web browser, in our case, the developed ICAP client creates a "forged" one,
- empty line (06), indicating end of HTTP request,
- HTTP response header (07-09). Again, forged by our client,
- empty line (10), indicating end of HTTP response header,
- the chunk size (line 11) as of [RFC2616, 3.6.1], which is a string of hex digits. The chunk size is here the same as the value specified in Context-Length of HTTP request,

---

<sup>6</sup>without chunk-extension

- the data (chunk-data), to be scanned (here the content of file test.txt) (line 12)
- the empty, last chunk, simply "0" (of course, followed by CRLF),
- empty line (CRLF) as of the Chunked-Body definition of [RFC2616, 3.6.1] (line 13).

As we've seen how encapsulating is done, let's get back to the "Encapsulated" header. It indicates an ICAP message that encapsulates a request header, a group of response headers, and then a response body. "The byte-offsets are in decimal notation for consistency with HTTP's Content-Length header [...] The order in which the encapsulated parts appear in the encapsulating message-body MUST be the same as the order in which the parts are named in the Encapsulated header." ([RFC3507, 4.4.1])<sup>7</sup>.

Basically, an ICAP client, which is used for scanning file on disk, must do the following steps before any data can be sent to the ICAP server.

- determine the file size,
- use this value for the "chunk-size" and "Content-Length" line,
- create "faked" HTTP response header,
- create "faked" HTTP request header,
- calculate size of HTTP request and response header,
- use those values for the Encapsulated header line,
- create ICAP request header.

### 3.2.3 ICAP response

An ICAP response could be

- "an error indication
- an encapsulated and potentially modified HTTP response header and response body
- an HTTP response 204 indicating that the ICAP client's request requires no adaptation" ([RFC3507, 4.9.2]).

The ICAP response "must start with an ICAP status line, similar in form to that used by HTTP, including the ICAP version and a status code" ([RFC3507, 4.3.3]). ICAP error codes, which differs from the HTTP counterparts are e.g:

---

<sup>7</sup>keep in mind, the byte-offset starts, of course, as position zero. And CLRF have to be taken into account.

204 - No modifications needed  
400 - Bad request  
404 - ICAP Service not found

Here's an example response to the above discussed request:

```
01: ICAP/1.0 200 OK
02: IStag: "1052324700"
03: Encapsulated: res-hdr=0, res-body=127
04:
05: HTTP/1.1 200 OK
06: Content-Type: application/octet-stream
07: Content-Length: 14
08: Via: 1.1 Symantec AntiVirus Scan Engine (ICAP)
09:
10: e
11: TEST TEST TEST
12: 0
13:
```

Line 01-13 is the complete ICAP response message, which consists of the ICAP header message (01-03) and an encapsulated HTTP response header (05-08) and the chunked data (10-13) (the response body).

### 3.2.4 Performance Consideration

The following three means should be considered to improve performance.

As seen in 3.2.3, the ICAP server sends the complete data back, even if it has not changed. When the client has cached the data, the data should not be send back to improve performance. To achieve this, "an ICAP client MAY include 'Allow: 204' in its request headers, indicating that the server MAY reply to the message with a '204 No Content' response if the object does not need modification" ([RFC3507, 4.6]). For illustration, here the example ICAP response:

```
ICAP/1.0 204 No Content Necessary
IStag: "1052324700"
```

Actually, the developed example ICAP client always sends the "Allow: 204" header. Furthermore, it expects an ICAP status code 204 when the content is virus-free (which violates the RFC, but I consider this OK for an example program).

For performance reasons, an ICAP client may cache ICAP responses (i.e. an HTTP proxy like Squid). ICAP uses the IStag ("ICAP Service Tag") response-header field, which could be considered as a "cookie". In the case of an anti-virus ICAP service, "the IStag might be a combination of the virus scanner's software version and the release number of its virus signature database. When the database is updated, the IStag can be changed to invalidate all previous

responses that had been certified as 'clean' and cached with the old IStag" ([RFC3507, 4.7]).

As explained in chapter 1, some file types do not need to be scanned. So, e.g. it does not make sense to transfer a huge AVI file over the network. Therefore, ICAP offers a method called "message preview" ([RFC3507, 4.5]). The icap client may send the first bytes of a file, the ICAP anti-virus server determines the file type, and may then report back to the client to send the rest of the file or no more data. It's (currently) not possible to send more than one preview request, i.e. the ICAP server can not response as "the first 30 bytes was not enough to detect file type, send me another 50 bytes" (see [FB2002] for details).

### 3.2.5 ICAP extensions

Normally, when ICAP is used for virus scanning of HTTP traffic, a virus notification is send back in HTML ("embedded" as HTTP response), which is then simply displayed by the used browser. But for the purposes for which we re-use the ICAP protocol, this would mean parsing the HTML response to get the virus name(s). As every ICAP server is very likely using it's own HTML file, a generic parsing approach is not possible. Therefore, the developed ICAP client uses header information which is sent by the Symantec AntiVirus engine, to get the desired information. The following headers are used, which have been specified officially later in the ICAP extensions draft ([MST2003]).

```
X-Infection-Found-Header = "X-Infection-Found: Type=" TypeID
                          "; Resolution=" ResolutionID
                          "; Threat=" ThreadDescription ";"
```

TypeID	= 0   1   2
ResolutionID	= 0   1   2
ThreadDescription	= TEXT

The TypeID is

- 0 virus infection
- 1 mail policy violation
- 2 container violations (e.g. decompression time has reached a specified time out)

ResolutionID is

- 0 file was not repaired
- 1 file was repaired
- 2 file should be blocked or rejected

ThreadDescription is a human-readable description of the threat, e.g. the virus name ([MST2003, 4.6]).

The developed icap-client uses the X-Infection-Found header for retrieving the virus name.

The X-Virus-ID can be used as an alternative to X-Infection-Found

```
X-Virus-ID-Header = "X-Virus-ID:" OneLineUSTEXT
OneLineUSText    = 1*( <any CHAR except CTLs> )

X-Violations-Found-Header = "X-Violations-Found:" count
1*( CR LF Filename CR LF ThreadDescription CR LF
ProblemID CR LF ResolutionID )

count            = 1*DIGIT
Filename         = TEXT
ThreadDescription = TEXT
ProblemID        = 1*DIGIT
ResolutionID     = 0 | 1 | 2
```

### 3.3 Example ICAP Client Implementation

For getting in touch with ICAP and especially in writing an ICAP client implementation for `samba-vscan`, the `"icap-client"` utility has been developed. It's a very basic and simple "proof-of-concept" implementation. It's written in C, to re-use some of the code later than for the `samba-vscan` package. It can be used for demonstrating the ICAP protocol, as it can be instructed to show the ICAP response generated by `"icap-client"`, the response (if any) send back by the ICAP server or both. Without any option (except for the file name), it simply displays whether a file is infected by a virus or not. Of course, in case of infection the virus name will be displayed, too. Therefore, `"icap-client"` can be use for scanning file(s) via an ICAP anti-virus scanning server. The program is, of course, licensed under the terms of the GNU General Public License (GPL) and comes with some source code documentation. For details about the implementation please refer to the section "Client Implementation Details", 3.3.2 (page 37).

#### 3.3.1 Usage of "icap-client"

```
icap-client file [-scr|-ssr|-sboth|-v|-h]
Getting help
icap-client -h
```

```
$ ./icap-client -h
ICAP example client for use with Symantec AntiVirus Engine 4.x
(C) by Rainer Link, 2003, OpenAntiVirus.org, \
<rainer@openantivirus.org>
```

```
Usage:
./icap-client file [-scr|-ssr|-sboth|-v|-h]
```

```
Purpose:
This program is a sample ICAP client implementation
```

which sends the specified file to an ICAP server to be scanned for viruses

Options:

```
file      the file to be scanned by ICAP server
-scr      show client request
-ssr      show server response
-sboth    show client request and server response
-v        verbose mode
-h        prints this help screen
```

This software is licensed under the terms of the GNU General Public License (GPL)

Here's an example output when scanning the file eicar.com, which is in fact the EICAR Test File virus file.

```
$ ./icap-client eicar.com -v
ICAP example client for use with Symantec AntiVirus Engine 4.x
(C) by Rainer Link, 2003, OpenAntiVirus.org, \
<rainer@openantivirus.org>
```

```
Open connection to ICAP server ...
Scanning file eicar.com ...
Sending headers to ICAP server ...
Sending file data to ICAP server ...
read data: 68
send data: 68
Retrieving response from ICAP server ...
Found ICAP response line ...
Scan result: File 'eicar.com' is infected with virus \
'EICAR Test String'
Closing connection to ICAP server ...
```

The "icap-client" program will connect only to an ICAP server running on localhost, port 1344 (the default ICAP port). To change this, please adjust

```
/* port ICAP server listens on as specified
   by ICAP protocol */
#define ICAP_PORT 1344
/* IP the ICAP server listens on */
#define ICAP_IP "127.0.0.1"
```

in the icap-client.c file and re-compile.

The program returns the following error values, so it can be used in shell scripts et al.

0 - all OK

- 1 - virus was found
- 2 - an error occurred

### 3.3.2 Client Implementation Details

As already mentioned, the "icap-client" can be used to determine whether a file is clean or infected, in the latter case the virus name is displayed. This program has been (mostly) developed for the Symantec AntiVirus Engine, as this product was the first ICAP server with anti-virus facility I received. To work properly, the ICAP server must fit the following two conditions:

#### **C1: the response must always consist only of the ICAP response header**

The "icap-client" program sends the

```
Allow: 204
```

in the RESPMOD request. The server **must** respond with

```
ICAP/x.y 204 <text>
[more header lines]
```

if the file is clean (1).

In the case of a virus infection, the ICAP server **must not** response with an ICAP body containing the cleaned file or a virus notification as HTML, but instead with an ICAP header **only** as

```
ICAP/x.y 403 <text>
[more header lines]
```

Actually, only Symantec AntiVirus Engine supports that by setting

```
ICAPResponse=0
ICAPActionPolicy=SCAN
```

in the symcscan.cfg configuration file (2). This can not be done via the ICAP protocol<sup>8</sup>. But this is too inflexible, as HTTP scanning requires to send data back, i.e. with this setting SAVE can't be used with squid-icap (which will be discussed later). I would suggest to introduce a "Allow: 403" line in the RESPMOD request header similar to the "Allow: 204" line.

#### **C2: the "X-Infection-Found" line must be sent back in case of virus infection**

In case of a virus infection, the "ICAP/x.y 403" response header **must** contain the following X-header line

```
X-Infection-Found: Type=ID1; Resolution=ID2; Threat=desc;
```

which is used to parse the virus name (3).

Let's have a closer look at those design issues:

<sup>8</sup>to be precise, the ActionPolicy can be changed via ICAP request according to Rui Ataide, i.e. "RESPMOD icap://host/avscan?action=SCAN"

(1) For speed reasons, we just do not need the same, unchanged data sent back from the ICAP server

(2)+(3) ICAP was actually designed as "a protocol aimed at providing simple object-based content vectoring for HTTP services" ([RFC3507]). For HTTP purpose, you can simply send back the virus information as an HTML page which is then displayed by the browser to inform the user. For other purposes, you must parse the information, but as every ICAP server uses its own HTML page/format, it's somewhat difficult to parse. In short, there's no generic way to do it currently, and therefore the "icap-client" has been designed as mentioned above to work (at least) with the Symantec AntiVirus Engine.

So, how could a generic way to get the virus name(s) look like? Here are three proposals

**P1:** "X-Infection-Found" line as a must for ICAP servers with anti-virus facility

An ICAP server with anti-virus facility must support the "X-Infection-Found" line in the response (regardless whether an additional ICAP body with an HTML virus notification page is sent or not). Moreover, it must be sent via an "ICAP/x.y 403" response<sup>9</sup> (or, an extra 4xx status code must be defined for this purpose). As normally the HTML virus notification page is send HTTP encapsulated via an "ICAP/x.y 200" header, the client must announce it supports the 403 status code within the request. This could simply be done via sending the "Allow: 403" line in the request.

**P2:** OPTIONS response must contain information how to retrieve the virus name(s)

The OPTIONS method is used to retrieve configuration information. Therefore, the OPTIONS response must contain information how to parse the virus name from the RESPMOD response.

**P3:** XML data in encapsulated HTTP body of the RESPMOD response

The servers sends an encapsulated HTTP response back, by which the HTTP body consists of XML data (instead of HTML). XML is used for structured information and can be easily parsed using some of the XML parsing libraries. The client must announce it expects XML data via the "X-XML-DATA: yes" (or similar) line in the request. The XML definition has to be specified in the ICAP protocol (or as extension).

Some more remarks on the implementation and drawbacks.

**I1:** It does not use Unix-I/O (i.e. read(2), write(2) function) but the Standard-I/O library instead as specified in the ANSI C standard. Actually, it opens an input and output stream on the opened socket (see [WRS2000, pp. 360]). It uses blocking I/O, which, simply speaking means that a process is blocked until an I/O system call returns. For this small utility this was considered as acceptable, but it's recommended to use other I/O techniques (e.g. I/O multiplexing or even better asynchronous I/O) (see [WRS2000, pp. 139]).

<sup>9</sup>actually, 403 means "forbidden" in HTTP/1.1, but here not the ICAP request itself was forbidden, but the content. So, defining an extra 4xx status code would be better

**I2:** It does not use any timeouts, for e.g. closing the connection when after 10 secs still no response from the ICAP server has been retrieved (see [WRS2000, pp. 343] for details).

**I3:** The file to be scanned is read in via an 8 kB buffer, which can be considered as a well-enough buffer.

**I4:** It doesn't read in the whole answer from the server, but aborts until it either has read enough to be able to recognize the file is clean (i.e. it aborts just after the "ICAP/x.y 204" line has been read) or file is infected and it has also retrieved and parsed the virus name.

**I5:** For simplicity, the message preview technique is not implemented. So, a file will always be transferred (completely), regardless if the file type of the transferred file is susceptible of a virus infection. Of course, this is a performance loss.

**I6:** The Content-Type in the "faked" HTTP-reponse is always set to application/octet-stream, so the actual file type is ignored. Therefore, the ICAP server must not trust this, but performing it's own file type detection. Actually, the file(1) utility can be used to determine the file type, but it does not provide an API for own programs (executing file(1) and grepping the output is not a real solution). The only C library currently known is FileType<sup>10</sup> from Paul L Daniels, but it's far beyond the reliability of file(1).

## 3.4 Performance

### 3.4.1 Hardware

The following hardware has been used for performance tests.

Name: rlss

Type: Laptop

CPU: Mobile Pentium III (Coppermine), 1GHz, 1998.84 bogomips

Memory: 256 MB SD-RAM

Hard Disc: HITACHI\_DK23CA-20, 20GB, 2048 kB cache, UDMA2, 20 MB/sec

NIC (chip): RealTek RTL-8139C, 100 Mb/s

OS: SuSE Linux 8.1

Kernel: 2.4.19-4GB (SuSE)

Filesystem: ext3

Java: Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1\_04-

b02) Java HotSpot(TM) Client VM (build 1.3.1\_04-b02, mixed mode)

Name: rlss2

Type: Midi Tower

CPU: AMD Duron, 600 MHz, 1199.30 bogomips

Memory: 256 MB SD-RAM

Hard Disc: IBM IC35L060AVVA07-0, 60GB, 1863 kB Cache, UDMA3, 32

MB/sec

NIC (chip): DC21143, 100Mb/s

<sup>10</sup><http://www.pldaniels.com/filetype/>

OS: SuSE Linux 8.0  
Kernel: 2.4.18-4GB (SuSE)  
Filesystem: ext3  
Java: Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.0) Classic VM (build 1.3.0, J2RE 1.3.0 IBM build cx130-20010626 (JIT enabled: jitc))  
Name: rlss3  
Type: Midi Tower  
CPU: Pentium II (Klamath), 266 MHz, 532.48 bogomips  
Memory: 192 MB SD-RAM  
Hard Disc: IBM-DJNA-351520, 30 GB, 430 kB Cache, UDMA2, 15 MB/sec  
NIC (chip): RealTek RTL-8029, 10Mb/s  
OS: SuSE Linux 6.2  
Kernel: 2.4.14  
Filesystem: ext2  
Note: this machine has only been used as web server for Squid tests

Connected with Cat-5 STP cable via W-LINX SW-008GC, 8 port 10/100Mb/s N-way switching hub.

### 3.4.2 Results

First of all, those results shouldn't be taken as absolute figures (i.e. if a program X is 5sec faster than Y, this doesn't count that much) but should give only some impression about performance (or the performance loss). The performance heavily depends on the speed of the scan engine. So, those results should be taken with a pinch of salt.

The following two tests have been done

- AVI file with size 1 MB, 10 MB, 100 MB and 500 MB. As **no** virus scanning is needed, the elapsed time for ICAP should be just the transfer time
- Transferring 6695 ELF files (566 MB) and 2220 Office files (517 MB). Those files are clean, the Office files did not contain any macros, as we don't want to time the scan engine.

All timings (elapsed real time between invocation and termination) have been done via "time(1)" command. Only the average time is shown here.

The tested programs and setups are outlined below. If possible, tests have been done via loopback interface (127.0.0.1) on rlss and rlss2, rlss → rlss2 and rlss2 → rlss via network (rlss → rlss2 means: ICAP server running on host rlss2, data is send from rlss to rlss2) .

- Symantec AntiVirus Engine (SAV), tested in ICAP mode and in NATIVE mode (AVI file test). Office and ELF test only in ICAP mode. Data send via the "icap-client" utility (for the Office + ELF test the "find(1L)" tool has been used, too)

- WebWasher CSM (WW). Data send via the "icap-client" utility (for the Office + ELF test the "find(1L)" tool has been used, too)
- for speed comparison, AVI file test has been done by copying the file.
- for speed comparison, AVI file test has been done with wget to retrieve the file(s) from the other host. Note, not the request direction but the data flow direction is mentioned in the table(s) and figures (i.e. rlss → rlss2 means data, the HTTP response, send from rlss to rlss2)

Product/Task	1M (s)	10M (s)	100M (s)	500M (s)
SAVSE (ICAP, local, rlss)	<1	<1	11	73
SAVSE (ICAP, local, rlss2)	<1	<1	7	54
SAVSE (ICAP, rlss → rlss2)	<1	<1	13	71
SAVSE (ICAP, rlss2 → rlss)	<1	1	13	80
SAVSE (NATIVE, local, rlss)	<1	<1	11	72
SAVSE (NATIVE, rlss → rlss2)	<1	1	13	70
SAVSE (NATIVE, local, rlss2)	<1	<1	7	53
SAVSE (NATIVE, rlss2 → rlss)	<1	1	13	77
WW (local, rlss)	<1	<1	11	80
WW (rlss → rlss2)	<1	1	15	73
WW (local, rlss2)	<1	<1	11	55
WW (rlss2 → rlss)	<1	1	11	68
copy (local, rlss)	<1	<1	12	77
copy (local, rlss2)	<1	<1	7	45
wget (rlss → rlss2)	<1	1	15	80
wget (rlss2 → rlss)	<1	1	14	72

Table 3.3: Performance results for AVI file test

Figure 3.7 illustrates the results for host rlss using WebWasher. Copying the 100MB/500 MB file took nearly the same time as sending the file via icap-client to WebWasher, running on host rlss (localhost). But even the elapsed time for sending it via icap-client to the remote host rlss2 (running WebWasher) is basically the same. But that's not really surprising as the hard disc of rlss is not that fast and by the local test, the HD must perform reads/writes, by the remote test only reads (the network throughput is around 6 MB/s). Of course, the results for host rlss2 (figure 3.8) differ. The faster hard disc results in better performance. And, the elapsed time of the local(host) tests are lower than for the remote test (data send to rlss, on which WebWasher runs).

As on the AVI file no virus scanning is performed, the results for Symantec AntiVirus Scan Engine are basically the same as for WebWasher (figure 3.9, figure 3.10).

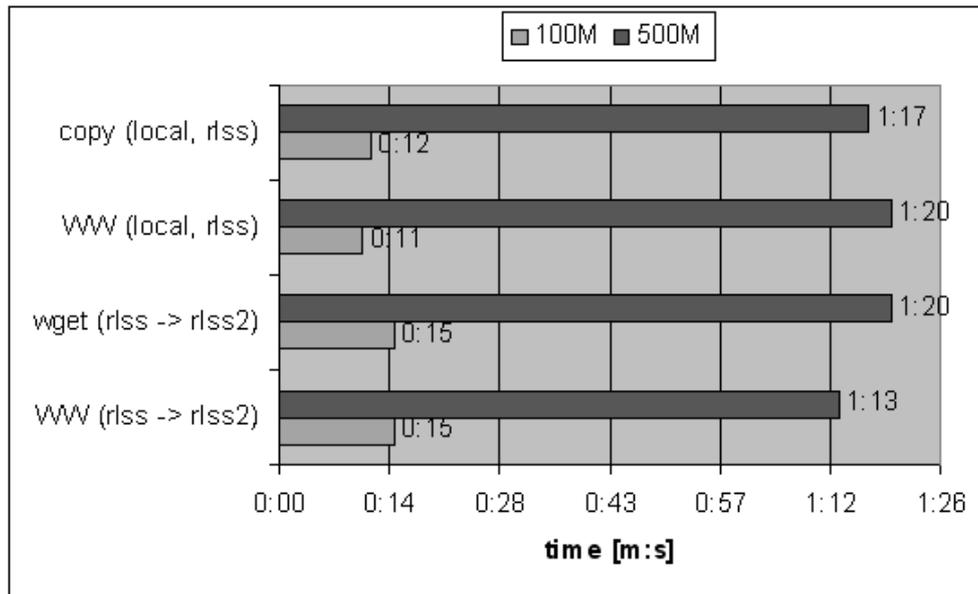


Figure 3.7: Performance results AVI file test, WebWasher, host r1ss

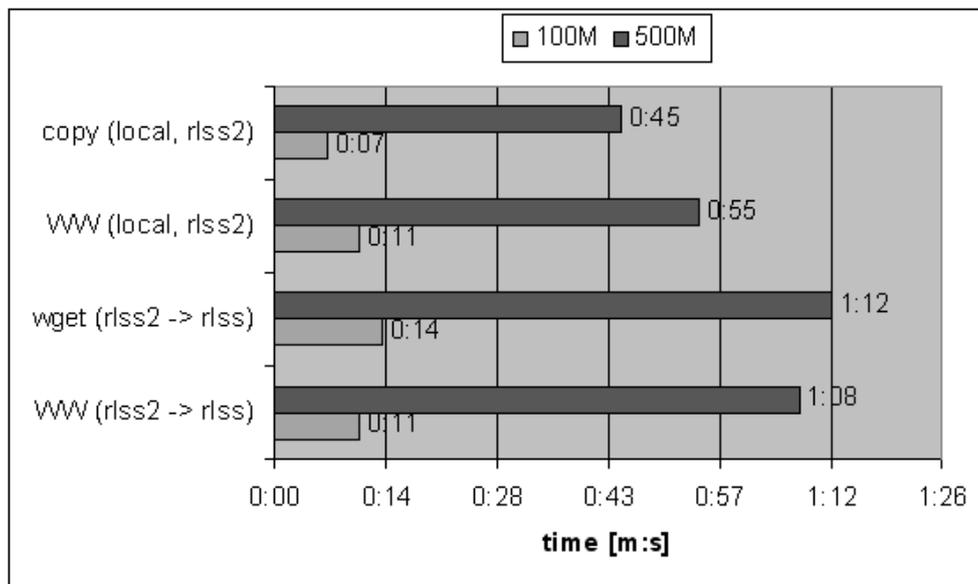


Figure 3.8: Performance results AVI file test, WebWasher, host r1ss2

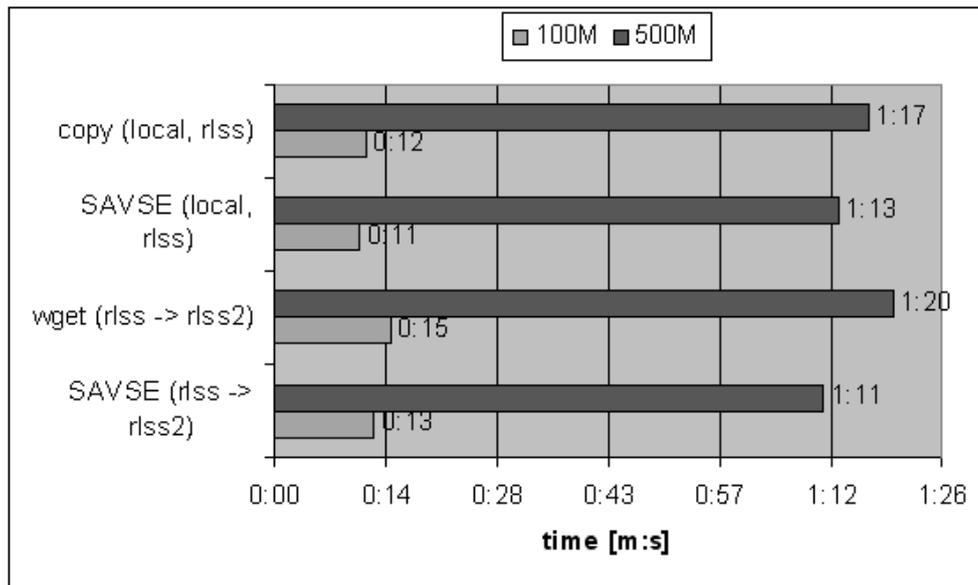


Figure 3.9: Performance results AVI file test, SAVSE, host r1ss

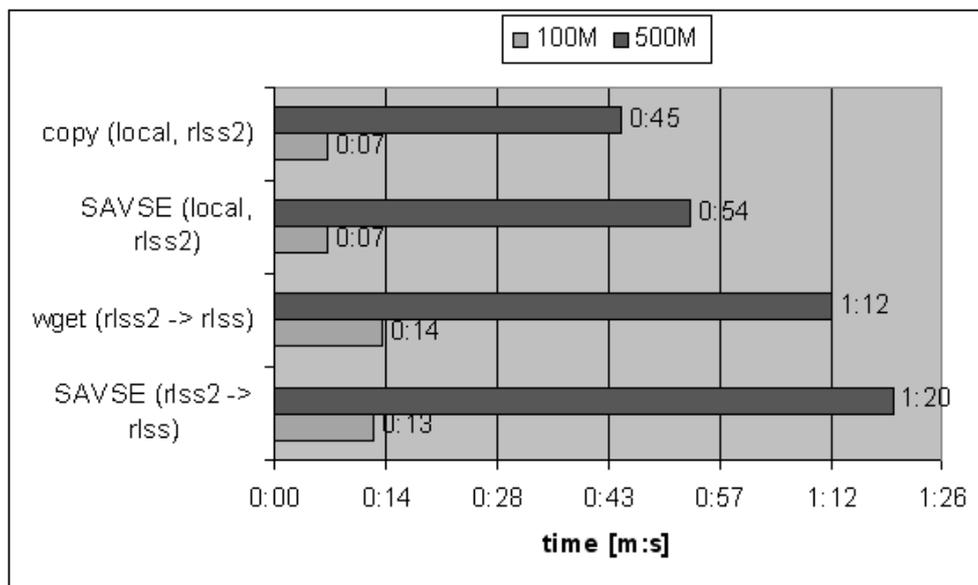


Figure 3.10: Performance results AVI file test, SAVSE, host r1ss2

Product/Task	ELF (m:s)	Office (m:s)
SAVSE (local, rlss)	2:59	1:53
SAVSE (local, rlss2)	3:25	1:23
SAVSE (rlss -> rlss2)	4:08	1:51
SAVSE (rlss2 -> rlss)	3:29	1:51
WW (local, rlss)	3:57	3:38
WW (local, rlss2)	4:24	3:33
WW (rlss -> rlss2)	4:45	3:32
WW (rlss2 -> rlss)	4:43	4:15
NAI AV	1:28	1:04
NAI AV (via find) <sup>11</sup>	-	23:28

Table 3.4: Performance Results for ELF/Office files test

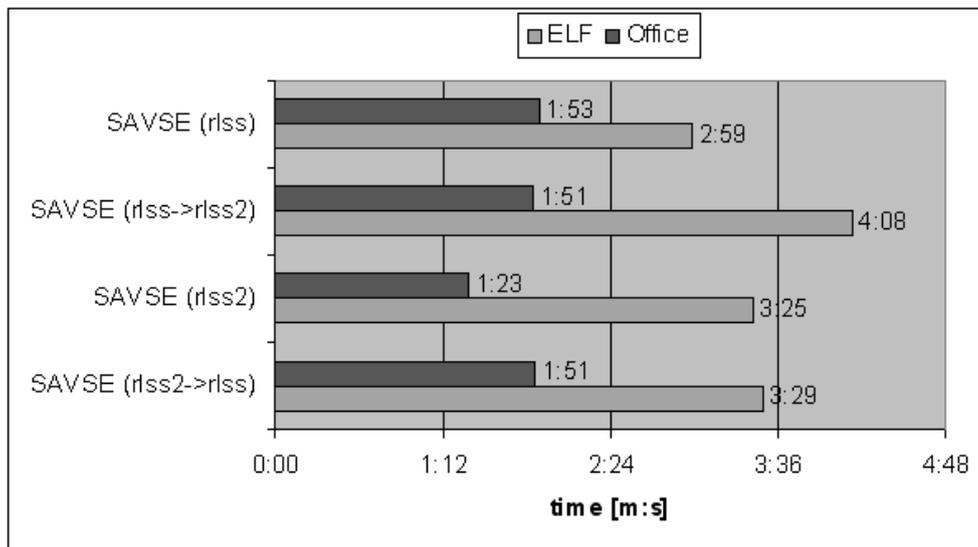


Figure 3.11: Performance results ELF/Office file test, SAVSE

As figure 3.11 for the Symantec AntiVirus Scan Engine illustrates, scanning Office files is faster on rlss2 (fast hard disc) than on rlss (slow hard disc). On the other hand, scanning ELF files is faster on rlss (faster CPU). Even sending the ELF files from rlss2 to rlss and then scanning on rlss requires (nearly) the same time as scanning locally on host rlss2. The slowest method for ELF files is sending them from host rlss to rlss2 to perform virus scanning on host rlss2.

The results for WebWasher (see figure 3.12) are in some cases the same but in some different. The elapsed time for the Office files test on host rlss and rlss2 are basically the same, so rlss2 is here not faster (as with SAVSE). Again, ELF files are scanned the fastest locally on host rlss, the slowest method is to send them from rlss to rlss2. But with WebWasher, sending ELF files from rlss2 to rlss for scanning is slower than scanning locally on host rlss2.

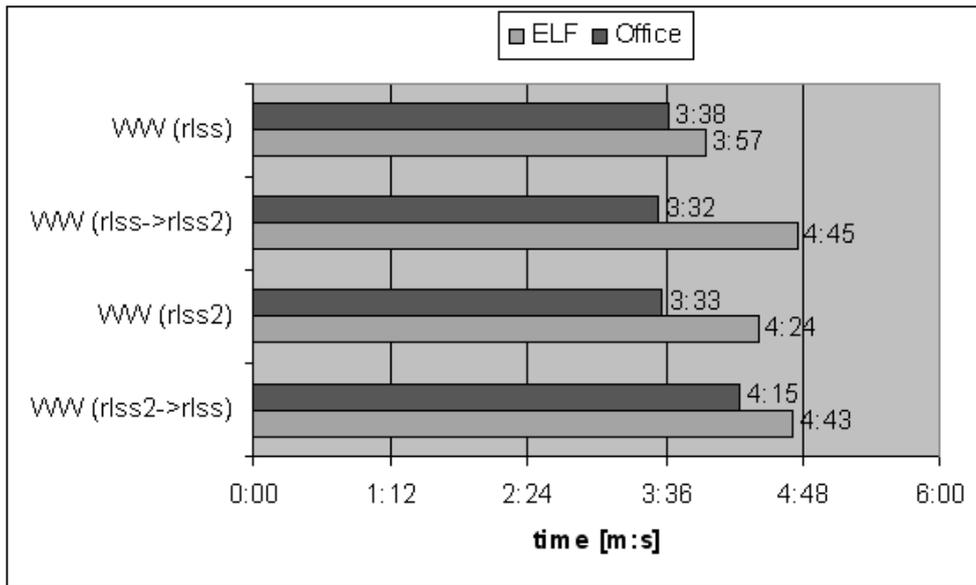


Figure 3.12: Performance results ELF/Office file test, WebWasher

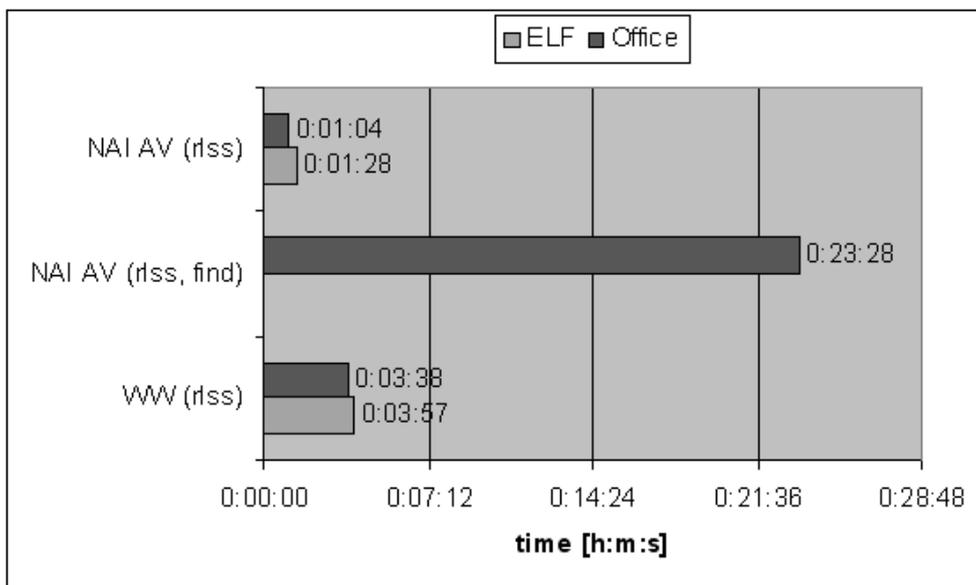


Figure 3.13: Performance results ELF/Office file test, NAI AV, WebWasher

The last figure 3.13 illustrates a comparison of NAI AntiVirus/Linux ("uvscan") and WebWasher (both use exactly the same engine and virus definition .DAT files). Of course, the on-demand virus scanner "uvscan" is faster when called as "uvscan -recursive /tmp/samba-test/office", than transferring all the files to WebWasher via ICAP, but much slower when called for each file via "find /tmp/samba-test/office -type f -exec uvscan {} \;"

### 3.5 Conclusion

On the one hand, most of the requirements (see 2.1) are met:

- ✓ ICAP is an open protocol, it's available for free as an RFC. Moreover, some ICAP server and/or client implementations are licensed under a Open Source/Free Software license, like the GNU Public License. Switching anti-virus vendors should be in most cases very easy, but the choice is limited as the list of available products is very short. If ICAP is used in a non-HTTP scenario, the ICAP client most likely needs some adaption for parsing the virus name(s).
- ✓ the implementation efforts can be considered as quite OK, as the protocol is not too complex. The maintenance efforts can not really be estimated (see below for details)
- ✓ load separation is possible, as well as load balancing (although it's not specified in the RFC itself)
- ✓ the performance should be quite fair for most cases
- ✗ easy integration (depends). Of course, a shell/perl script is able to call an external program, like the example "icap-client", but calling an external program may be slow and gives the control to it. A Perl ICAP module would be handy<sup>12</sup>.

But on the other hand, I would question the future of ICAP. The traffic on the ICAP-discussion mailing list, to which I'm subscribed since the beginning of this thesis, is very low<sup>13</sup>. The IETF is currently working on Open Pluggable Edge Services (OPES)<sup>14</sup> which considers ICAP as a precursor: "The Open Pluggable Edge Services (OPES) working group is chartered to define a framework and protocols to both authorize and invoke distributed application services while maintaining the network's robustness and end-to-end data integrity. These services may be server-centric (i.e., an administrative domain that includes the origin server) and they may be client-centric (i.e., an administrative domain that includes the user agent)" ([OPS03]). The virus scanning scenario is mentioned in [ABA2002, section 3.2]. Currently, the IETF WG

---

<sup>12</sup>search.cpan.org will show some hits for the keyword ICAP, but those modules implement or use the Internet Calendaring Protocol

<sup>13</sup>see the monthly statistics at <http://groups.yahoo.com/group/icap-discussions/>

<sup>14</sup><http://www.ietf-opes.org/>

is discussing the OPES call-out protocol (OCP), which supports the remote execution of OPES services. As especially the OCP specification is incomplete, it's currently not possible to estimate efforts for switching from ICAP to OPES/OCP and how the migration path would look like.



# Chapter 4

## Mailserver - AMaViS

This chapter will give a brief overview about the email scanning tool "A Mail Virus Scanner" (AMaViS), which can be used to integrate several virus scanners into the four most common Mail Transfer Agents (MTA). How this can be done is demonstrated with the MTA sendmail and postfix.

### 4.1 MTA-Integration

In this section, the AMaViS integration into sendmail and postfix will be discussed. Those concepts are of course valid for other tools as well and may be used for other purposes (i.e. general content filtering), too.

#### 4.1.1 Sendmail

Basically, AMaViS can be integrated into sendmail via three ways:

- by replacing the settings for the local delivery agent (LDA)
- by using a dual sendmail setup with different sendmail configuration
- by using sendmail's content filtering API

The first two methods ([RL2003]) are not recommended: the LDA method can by design only scan incoming mails, the dual sendmail method is complicated to set up. The last one is preferred, and will be outlined here.

The **sendmail content filtering API**, mostly referred as (lib)milter interface, "provides third-party programs to access mail messages as they are being processed by the Mail Transfer Agent (MTA), allowing them to examine and modify message content and meta-information. [...] Filters can process messages' connection (IP) information, envelope protocol elements, message headers, and/or message body contents, and modify a message's recipients, headers, and body." ([SI2000]). Such filters run as separate processes for security and stability reasons and communicate with the MTA via local or remote connections, i.e. Unix domain or TCP sockets. "The Milter library (libmilter) implements the communication protocol. It accepts connections from various MTAs, passes the relevant data to the filter through callbacks, then makes

appropriate responses based on return codes. A filter may also send data to the MTA as a result of library calls.” ([SI2000]). Table 4.1 shows the milter callbacks related to an SMTP transaction, Figure 4.1 illustrates how milter works.

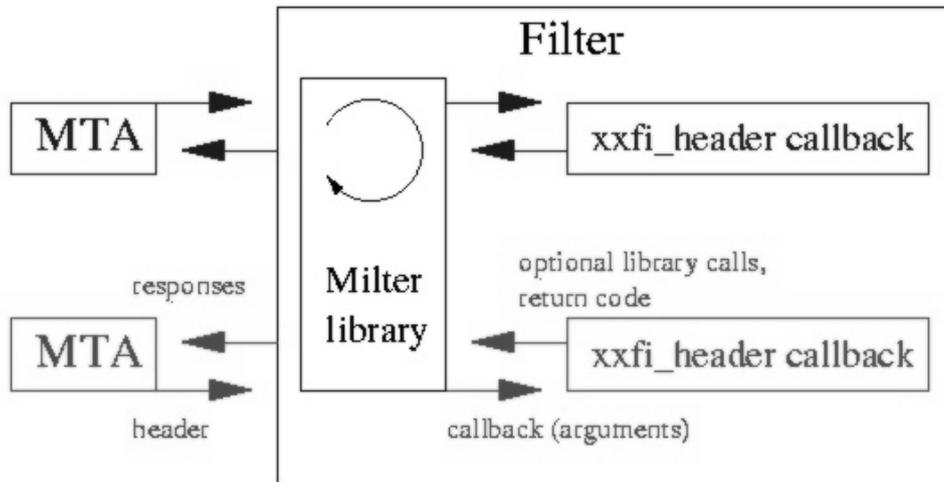


Figure 4.1: Sendmail: A filter handling simultaneous requests from two MTA’s ([SI2000])

SMTP commands	Milter callbacks
(open SMTP connection)	xxfi_connect
HELO ...	xxfi_helo
MAIL FROM ...	xxfi_enfrom
RCPT TO ...	xxfi_envrcpt
DATA	
header ...	xxfi_header
	xxfi_eoh
body ...	xxfi_body
.	xxfi_eom
QUIT	xxfi_close
(close SMTP connection)	

Table 4.1: Milter callbacks related to SMTP transaction ([SI2000])

The communication between AMaViS and sendmail is done via a Unix Domain socket (on SuSE Linux this is /var/run/amavis/amavis-milter.sock). The milter has to be started before sendmail, e.g. on SuSE Linux (that’s actually a one-liner)

```
startproc -u vscan /usr/sbin/amavis-milter \
-p local:/var/run/amavis/amavis-milter.sock \
>/dev/null 2>&1 &
```

Of course, sendmail must be configured to use this filter. The following m4<sup>1</sup> code is used to set it up.

```
define('MILTER')dnl
divert(-1)
INPUT_MAIL_FILTER('milter-amavis', \
'S=local:/var/run/amavis/amavis-milter.sock, \
T=S:10m;R:10m;E:10m')
divert(0)dnl
```

This will result (shortened) to the following code in sendmail.cf

```
# Input mail filters
0 InputMailFilters=milter-amavis

Xmilter-amavis, \
S=local:/var/run/amavis/amavis-milter.sock, \
T=S:10m;R:10m;E:10m
```

#### 4.1.2 Postfix

Postfix (sic!) Content Filtering mechanism differs from the sendmail one: "A Postfix content filter receives unfiltered mail from Postfix and either bounces the mail or re-injects filtered mail back into Postfix" ([WV2003]). So, basically, postfix sends the mail to be checked to a filter, either via pipe transport or via LMTP/SMTP. The filter may then deliver back the mail either by piping it to the sendmail wrapper program or sending it back via SMTP ([WV2003]).

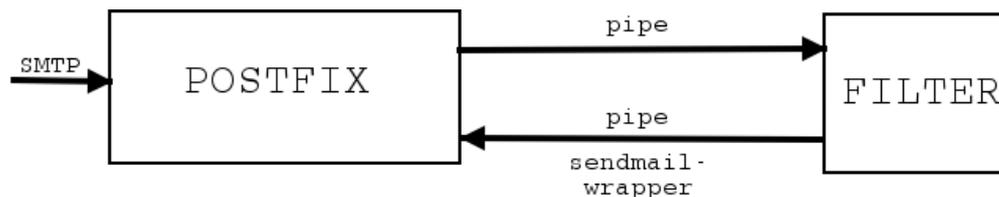


Figure 4.2: Postfix: simple content filtering example, based upon [WV2003] (simplified)

Figure 4.2 shows a simple content filtering example, by which mail is sent via the pipe transport to the filter and the filter pipes the mail eventually back to the sendmail wrapper (in fact, postfix consists of several programs doing specific tasks instead of the monolith original sendmail program).

<sup>1</sup>a macro processor, see <http://www.gnu.org/software/m4/m4.html> and [http://www.sendmail.org/m4/intro\\_m4.html](http://www.sendmail.org/m4/intro_m4.html)

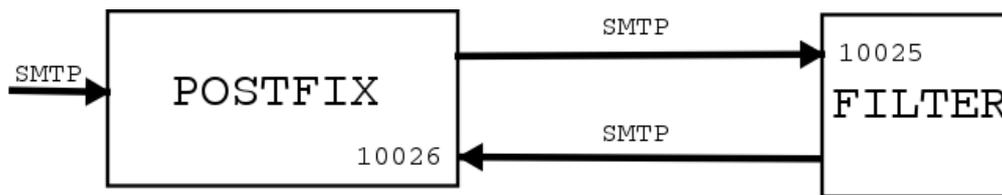


Figure 4.3: Postfix: advanced content filtering example, based upon [WV2003] (simplified)

Figure 4.3 shows the so called advanced content filtering example. In this case, mail is send via SMTP to port 10025, on which the filter listens on. The mail may be delivered back to postfix via SMTP, listening on port 10026.

The AMaViS setup is actually a mixture of both: the mail is piped to the AMaViS perl script, and AMaViS delivers the mail back via SMTP to postfix.

The following changes are therefore needed to the postfix configuration.

For `/etc/postfix/main.cf`:

```
content_filter = vscan:
```

For `/etc/postfix/master.cf`:

```
vscan      unix - n n - 10 pipe \
user=amavis argv=/usr/sbin/amavis ${sender} ${recipient}

localhost:10025 inet n - n - - smtpd \
-o content_filter=
```

## 4.2 Features

As already mentioned, AMaViS supports the probably most-commonly used Mail Transfer Agents and a wide range of anti-virus products (each virus scanner has it's own "module" in the `amavis/av` subdirectory) . As not all of them include MIME/UUENCODE support or are able to handle all those various compression formats, AMaViS is responsible for such tasks. So, basically, AMaViS receives the complete mail from the MTA and stores it onto disk (e.g. as `/var/amavis/amavis-<uniquename>/email.txt`). Now, this mail will be splitted into its parts, i.e. the mail itself and the attachment file(s) (if any), and stored on disk (e.g. in directory `/var/amavis-<uniquename>/parts/`). For each of those files, the file type will be detected<sup>2</sup>, and if it's compressed/archived, the corresponding decompression/unarchiving tool will be run. As last step, the configured virus scanner(s) is/are called. Depending on e.g. the return value or the output of the scanner AMaViS determines whether an email (attachment) was infected or not. In case of an infection, the mail will be moved into quarantine (and therefore delivery stopped) and, by default, a virus notification is sent to the sender of the email message and the administrator of the mail server. To add ICAP support to AMaViS, simply a new module has been created, which

<sup>2</sup>via the Unix-command `file(1)`; not via the file extension, which is just too unreliable

simply calls the "icap-client" utility to send the file(s) to the ICAP server for scanning.

### 4.3 Conclusion

In the AMaViS case, ICAP is simply re-used for on-demand scanning files (as mentioned above, AMaViS splits the mails in it's parts and writes them to disk) using the "icap-client" utility. Therefore, no specific performance tests have been run (please see 3.4.2, p. 40 for on-demand scanning tests). Of course, calling the external "icap-client" program is slower than doing the ICAP communication completely in Perl. But for evaluation purposes, I did not consider this as important and decided to not write a Perl ICAP module. Actually, AMaViS + ICAP worked stable and may used to off-load (only) the virus scanning process to another host, i.e. MTA and AMaViS running on host X, ICAP anti-virus facility on host Y. For high-loaded mail servers, it's probably a good idea to off-load not only the virus scanning process but AMaViS completely, i.e. MTA on host X, AMaViS + ICAP server on host Y (or AMaViS on host Y, ICAP server on host Z).



# Chapter 5

## Fileserver - samba-vscan

Samba is an Open Source implementation of the SMB<sup>1</sup>/CIFS<sup>2</sup> standard, used by Microsoft network(s). Samba allows seamless integration of Linux/Unix file servers into a Microsoft network, i.e. Windows (file) server(s) and clients. Generally speaking, Samba offers file-, print- and authentication services ([CH2001]). Samba is one of the most popular open-source projects.

Basically two concepts are possible for protection the file server, which will be discussed in this chapter.

### 5.1 Concepts

#### 5.1.1 Not Integrated into Samba

The easiest method is to set-up a cron job, which will scan all shared directories periodically. But this leads to a "time gap", in which an infected word document can be placed on a share, without any notice. This document may have been opened several times until the next cron job run starts.

Therefore, a better way is on-access scanning, i.e. to scan a file while it's being opened and/or closed. Assuming, the Samba servers runs under the Linux OS, this can be done by "hooking" system calls for opening or closing a file to perform a virus scan first and deny access to the file upon infection. For this purpose, the kernel sources could be changed or a kernel module could be written. The first method requires a re-build of the kernel and patching the kernel for every new kernel release which is somewhat clumsy. The more elegant way is writing a kernel module, which will be explained briefly. The basic concept as outlined below is taken from the Dazuko project ([JO2003]). The name Dazuko comes from the german word "**D**ate**z**ugriff**s**kont**r**olle", which means file access control.

"A module is an object file whose code can be linked to (and unlinked) from the kernel at runtime. The object code usually consists of a set of functions that implements a filesystem, a device driver, or other features at the kernel's upper layer." A module "[..]" is executed in Kernel Mode on behalf of the

---

<sup>1</sup>Server Message Block, for details see ([RS2002])

<sup>2</sup>Common Internet File System, for details see ([CRH2003])

current process, like any other statically linked kernel function" ([DBMC2001, p. 11]). So, the kernel module extends/changes the functionality of e.g. the "open" call. As Linux supports a wide range of file systems, you may assume the "open" call functions for each of this file systems has to be changed. But, actually, since Linux supports a concept called as "Virtual Filesystem" (aka "Virtual Filesystem Switch" or VFS), this is not needed. The VFS "is a kernel software layer that handles all system calls related to a standard Unix filesystem. Its main strength is providing a common interface to several kinds of filesystems" ([DBMC2001, p. 328]). Or, in other words, "the key idea behind the VFS consists of introducing a common file model capable of representing all supported filesystems. This model strictly mirrors the file model provided by the traditional Unix filesystem". "The first Virtual Filesystem was included in Sun Microsystem's SunOS in 1986" ([DBMC2001, p. 330]). The actual virus scanning isn't done in kernel space, of course, but in userland instead. So, e.g. a running anti-virus daemon communicates via a (character) device (e.g. /dev/dazuko for Dazuko), e.g. for sending the filename to be scanned from module to daemon and then the result back. In case of an infection, the module "tells" the kernel to not allow access. Obviously, this method can not be used only for scanning Samba shares, but e.g. an ftp incoming directory.

Here's a brief demonstration of Dazuko. First, the system hasn't loaded the on-access scanning facility. Therefore, a simple cat(1) on the file eicar.com is possible.

```
# cat eicar.com
```

```
X50!P%@AP[4\PZX54(P^)7CC7}$EICAR-STANDARD-\
ANTIVIRUS-TEST-FILE!$H+H*
```

Now, the service is being started

```
# rcavguard start
```

```
Starting AntiVir on-access scanning facility: avguard
loading dazuko kernel module
creating character device
starting avguard
```

And cat(1) is no longer able to print the contents of the same file.

```
# cat eicar.com
```

```
cat: eicar.com: Operation not permitted
```

Finally, the corresponding log snippet from /var/log/messages

```
rlss insmod: Using /lib/modules/2.4.19-4GB/misc/dazuko.o
rlss kernel: Dazuko: loaded, version=1.1.2, dev_major=253
rlss antivir[5610]: AntiVir ALERT: \
[Eicar-Test-Signatur virus] /tmp/eicar.com \
<<< Contains code of the Eicar-Test-Signatur virus
```

### 5.1.2 Integrated into Samba

On-access scanning, integrated into Samba, can be achieved via the Samba VFS switch, which was originally written as a patch for Samba 2.0 by Tim Potter. It can be used to extend the regular Samba file system, i.e. features which are lacking from the underlying Unix file system. "For example, you might want a Mac-like 'trash can' where deleted files go, to give you a chance to recover for accidental deletions. Or, you might want to translate between Unix and DOS line-ending conventions on recognized kinds of text file. [...] In general, using a VFS is an elegant way to add features to a filesystem, without having to know everything about the filesystem implementation" ([DCB2000]).

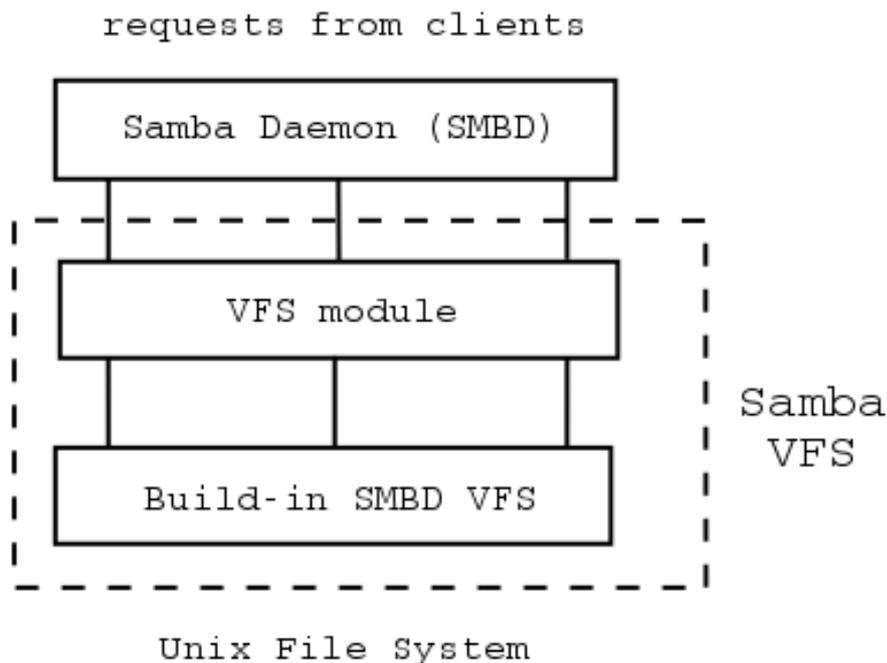


Figure 5.1: Samba VFS (simplified), based on [AB2002]

So, to some extent similar to the kernel VFS as mentioned in the previous section, Samba VFS can be used in a so-called VFS module to implement on-access virus scanning by hooking Samba's default "open" and "close" call for example. Such a module will be dynamically loaded via `dlopen`.

Due to the limitation of the VFS layer in Samba 2.2.x, it is not possible to use more than one VFS module per share. This could be considered as a major drawback as it may be desirable to offer an on-access scanning and a recycle bin facility (similar to the Windows recycle bin). This limitation does not exist in Samba 3.x (in recent alpha versions as of date of this writing), due to the cascading VFS module interface (also known as stackable VFS). VFS of Samba 3.0 has several operation layers, but only the opaque and transparent layer are differentiated. The opaque layer should be used by modules which implement actual file system, mainly for implementing a new file system (like

DB-based VFS). An operation marked as opaque is the final level, i.e. it does not call anything beyond itself. An operation marked as transparent will call underlying layer later. The other layer types (audit, splitter, scanner) were designed to provide different degree of transparency and for diagnosing VFS module behaviour (taken from comments of the file source/include/vfs.h of the Samba 3.0 source code). Good documentation is currently lacking and the VFS interface has changed from time to time and will most likely change now and then<sup>3</sup>. For details, the samba-technical mailing list is a good resource; and, of course, the Samba source code. It's to be expected an extended VFS developers chapter will be added to the Samba Developers Guide soon.

Using a VFS module for the share "test" can be configured very easily with the following setting in the Samba configuration file smb.conf:

```
[test]
  path = /home/test
  vfs object = /usr/local/samba/lib/vfs/vscan-icap.so
  vfs options = config-file = /etc/samba/vscan-icap.conf
```

Here, the module vscan-icap.so (part of the samba-vscan package, see below) will be used for on-access scanning ("vfs object" setting). The "vfs options" setting specifies the configuration file.

### 5.1.3 Pros & Cons

Of course, each concept has it's pros and cons.

kernel-based:

- + multi-purpose, not limited to Samba
- limited to special Unix-OS and/or versions (i.e. only for Linux)
- sending a message via Windows Messaging (winpopup) to inform the user about the virus being detected is not possible

Samba-integrated:

- + works with lot of Unices (e.g Linux, \*BSD, HP-UX, Solaris, AIX)
- + sending message via Windows Messaging is possible
- limited to Samba

## 5.2 Requirements

As samba-vscan should provide the same features as similar anti-virus products on Windows servers, the requirements are:

### **R1: block access of infected files**

This is achieved by hooking the "open" / "close" call in the Samba VFS layer. So, depending on the configuration, a file is scanned while it's opened or closed;

<sup>3</sup>Samba 3.0 is currently alpha code, so changes are just normal.

or both. If a file is infected, it will be blocked by not calling the underlying "open" / "close" function, but by returning -1 and setting errno to EACCESS. The user will see an access denied dialog box (see figure 5.2).



Figure 5.2: Samba-vscan: access denied

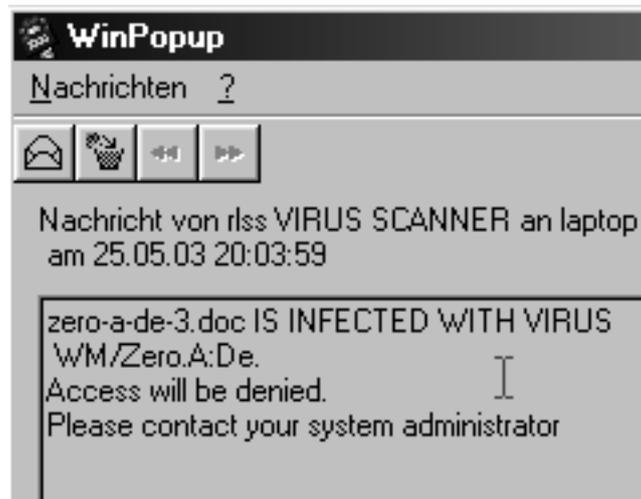


Figure 5.3: Samba-vscan: virus notification

### **R2: move infected files into quarantine / delete file**

If configured, an infected file will be moved into a special quarantine directory, or deleted. Per default, an infected file is left untouched. Quarantine is done by renaming the existing file; as the Samba VFS function "rename" is used, even renaming across filesystems works<sup>4</sup> (as the file will be copied in this case, see function `copy_reg()` and `vfswrap_rename()` in `source/smbd/vfs-wrap.c` (Samba 2.2.8 source code)).

### **R3: notification message**

When access to a file is being blocked, Windows will display the "Access denied" message box. As this text is always the same and can not be changed, other means to give more details to the user are required. If enabled, a notification

<sup>4</sup>`rename(2)` would return -1 and set errno to EXDEV in such as case

message via the Windows Messaging Service<sup>5</sup> will be sent. On Win9x, the client application "winpopup" must be running; on other Windows versions this service must be enabled. The code tries not to flood the user with messages, i.e. if the user clicks on the same file again, no notification will be sent. As, according to [MS2002], the popup queue can only receive six messages, the "notify only once" method tries to avoid that important (other) messages may get lost otherwise. See screenshot 5.3 as an example.

The administrator won't get a notification via this method. As samba-vscan logs (see R4) all it's activities via the syslog facility, a log watcher program can be used to match on alert messages, which can be send then via email, pager or SMS to the administrator.

Other requirements are:

#### **R4: logging**

All activities are logged via the syslog facility. Normal, informational messages are logged with they keyword "INFO", error messages with "ERROR" and virus alerts with "ALERT". The syslog facility (here: subsystem, the type of program that is logging the message, e.g. mail) and syslog priority<sup>6</sup> are currently compile-time settings (see include/vscan-global.h of samba-vscan source).

#### **R5: configuration**

samba-vscan provides a Samba-style configuration file for each virus scanner it supports. These settings include general options, e.g. if verbose file logging should be switched on or if file exceeding a specific file size it should be skipped. As well as anti-virus product related settings, i.e. on which port the anti-virus daemon is listening.

#### **R6: multiple scanner support**

The software ships with a own (VFS) module for each virus scanner being supported:

- vscan-fprotd: for use with F-Prot Daemon
- vscan-icap: for use with an ICAP anti-virus service, currently only Symantec AntiVirus Engine is fully supported (see 3.3.2, page 37 for details)
- vscan-kavp: for use with Kaspersky AntiVirus (kavdaemon)
- vscan-mks: for use with mks32
- vscan-oav: for use with OpenAntiVirus ScannerDaemon
- vscan-sophos: for use with Sophos Sweep via Sophie
- vscan-trend: for use with Trend Micro FileScanner via Trophie

#### **R7: modular design / framework**

As the modules offer basically the same functionality, most functions / facilities can be provided via a framework. This includes the needed VFS definitions, the

---

<sup>5</sup>see Microsoft Knowledge Base Article - 168893 at <http://support.microsoft.com/default.aspx?scid=kb;en-us;Q168893> for details

<sup>6</sup>see man 5 syslog.conf and man 3 syslog

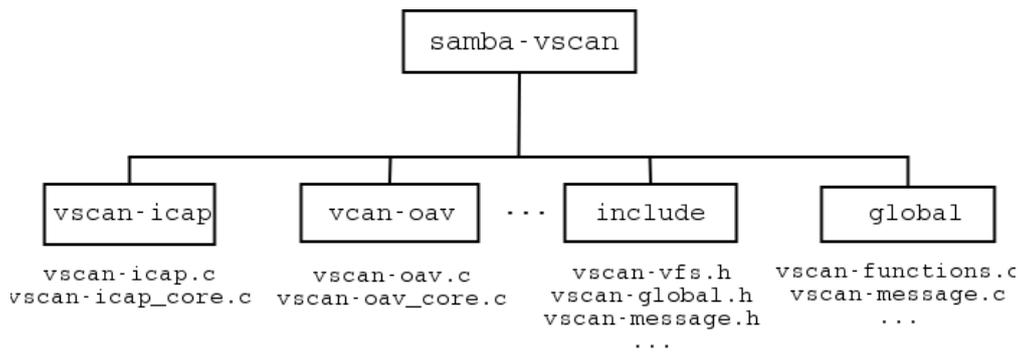


Figure 5.4: Samba-vscan: architecture

quarantine & windows messaging stuff and last recently accessed file mechanism (see R8). As well as some various other functions (i.e. logging). The parsing of configuration file is currently implemented by each module, but as most options are the same for each of them, this functionality could be provided by the framework, too. Or, in other words, the framework could and should be extended in future versions. The general architecture of samba-vscan is illustrated by figure 5.4. The code of each supported virus scanner is located in a corresponding sub directory. Definitions, constants and alike, used by each module, is located in header files in the "include" directory. Useful smaller functions (i.e. logging) are provided by the framework, located in "global". Mechanism, like quarantining and sending notification messages are provided as a "black box", too.

#### **R8: minimize the performance impact**

Obviously, virus-scanning has an impact on performance. During development the following two issues have been revealed.

**I1:** At least Win9x handles connections to shares via one socket connection to Samba (only one smbd process will be forked and handling multiple shares). On Linux/Unix, the Samba client utility "smbclient" opens a socket connection to Samba for each share (i.e. n connections for n shares). So, on Linux, if scanning of file on one share is very time consuming, this does not effect the performance of the other shares. On Win9x systems, this will slow down the speed/responsiveness of all shares. Unfortunately, there's no solution or even any workaround known ([MP2001]).

**I2:** Windows may open a file several times in a very short period of time. Here some examples: if a directory is listed, Windows opens some files to check their file type; or if it's an executable file, it will be opened to load the icon (so that this icon can be displayed in the listing). Or, if Samba reports back "access denied", Windows will try to open it up to six times, until Windows gives up and shows the "access denied" dialog box. This all happens usually in a time frame below one second. But as the "open" call is hooked, this results into scanning a file multiple times, whereas once would be sufficient. To solve this, the last recently accessed file(s) mechanism has been implemented. Per default,

it stores 100 entries for 5 seconds (both is configurable). Each entry contains the file name, the modified time of the file, the time the entry has been added and a flag indicating whether this file is infected or not. Those entries are stored in a double-linked list<sup>7</sup>. So, let's assume, the file "test.exe" is opened first. The mechanism detects, it's not in the list, so it has to be scanned. The result of the scan (here: clean) will then be stored. On the second open, the access will be granted without a scan (assumed, the file has not been changed). If the file would have been infected, the access would be denied without a new scan process. If the second access to the file happens later than five seconds, the entry will be regarded as invalid. This mechanism has been proven effective up to now.

This mechanism could be even used to reduce the need for scanning a file as long it's not being changed. But the entries are not stored permanently, i.e. when the user disconnects from the shares, all entries will be destroyed.

```

May 14 12:41:54 r1ss smbd_vscan_icap[5267]: INFO: Scanning file : '/tmp/samba-test/office/office1/office/office1/virensb1.ppt'
May 14 12:41:54 r1ss smbd_vscan_icap[5267]: INFO: file /tmp/samba-test/office/office1/office/office1/virensb1.ppt is clean
May 14 12:41:54 r1ss smbd_vscan_icap[5267]: File '/tmp/samba-test/office/office1/office/office1/virensb1.ppt' has already been scanned, not marked as infected and not modified. Not scanned anymore. Access granted
May 14 12:41:54 r1ss last message repeated 2 times

```

Figure 5.5: Samba-vscan: single-click on PowerPoint file

Figure 5.5 shows the log snippet, after a user did a single click on a power point presentation (using Explorer). As the Explorer is configured to show a short "preview" of files, the file is opened four times. The LRU mechanism avoids the second, third and fourth scan and grants access immediately.

<sup>7</sup>a single-linked list would be sufficient, but Samba offers DLIST\_\* macros for implementing a double-linked list

```
May 14 12:53:03 r1ss smbd_vscan_fprotd[5413]: samba-vscan (vscan-fprotd 0.3.3beta2) loaded (Samba >=2.2.4), (c) by Rainer Link, OpenAntiVirus.org
May 14 12:53:03 r1ss smbd_vscan_fprotd[5413]: INFO: connect to service test by user nobody
May 14 12:53:13 r1ss smbd_vscan_fprotd[5413]: INFO: Scanning file : '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC'
May 14 12:53:13 r1ss smbd_vscan_fprotd[5413]: ALERT - Scan result: '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' infected with virus 'WM/ABC.A', client: '192.168.0.10'
May 14 12:53:13 r1ss smbd_vscan_fprotd[5413]: File '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' has already been scanned and marked as infected. Not scanned any more. Access denied
May 14 12:53:14 r1ss last message repeated 2 times
```

Figure 5.6: Samba-vscan: double click on infected Word file I

```
May 14 12:56:22 r1ss smbd_vscan_fprotd[5486]: samba-vscan (vscan-fprotd 0.3.3beta2) loaded (Samba >=2.2.4), (c) by Rainer Link, OpenAntiVirus.org
May 14 12:56:22 r1ss smbd_vscan_fprotd[5486]: INFO: connect to service test by user nobody
May 14 12:56:30 r1ss smbd_vscan_fprotd[5486]: INFO: Scanning file : '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: ALERT - Scan result: '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' infected with virus 'WM/ABC.A', client: '192.168.0.10'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: INFO: Scanning file : '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: ALERT - Scan result: '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' infected with virus 'WM/ABC.A', client: '192.168.0.10'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: INFO: Scanning file : '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: ALERT - Scan result: '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' infected with virus 'WM/ABC.A', client: '192.168.0.10'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: INFO: Scanning file : '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC'
May 14 12:56:31 r1ss smbd_vscan_fprotd[5486]: ALERT - Scan result: '/tmp/makro-viren/MAKRO-SO/ABC-A.DOC' infected with virus 'WM/ABC.A', client: '192.168.0.10'
```

Figure 5.7: Samba-vscan: double click on infected Word file II

```
fprotd[5487]: Scanner arguments: -dumb -archive
fprotd[5487]: Scanning file: /tmp/makro-viren/MAKRO-SO/ABC-A.DOC
fprotd[5487]: Detected: WM/ABC.A
/tmp/makro-viren/MAKRO-SO/ABC-A.DOC Infection: WM/ABC.A
fprotd[5487]: File was infected.
fprotd[5488]: Scanner arguments: -dumb -archive
fprotd[5488]: Scanning file: /tmp/makro-viren/MAKRO-SO/ABC-A.DOC
fprotd[5488]: Detected: WM/ABC.A
/tmp/makro-viren/MAKRO-SO/ABC-A.DOC Infection: WM/ABC.A
fprotd[5488]: File was infected.
fprotd[5489]: Scanner arguments: -dumb -archive
fprotd[5489]: Scanning file: /tmp/makro-viren/MAKRO-SO/ABC-A.DOC
fprotd[5489]: Detected: WM/ABC.A
/tmp/makro-viren/MAKRO-SO/ABC-A.DOC Infection: WM/ABC.A
fprotd[5489]: File was infected.
fprotd[5490]: Scanner arguments: -dumb -archive
fprotd[5490]: Scanning file: /tmp/makro-viren/MAKRO-SO/ABC-A.DOC
fprotd[5490]: Detected: WM/ABC.A
/tmp/makro-viren/MAKRO-SO/ABC-A.DOC Infection: WM/ABC.A
fprotd[5490]: File was infected.
```

Figure 5.8: F-Protd: double click on infected Word file

Figure 5.6 shows the log snippet of a double-click of a infected Word document. Windows98 tries to open it four times (the ALERT line is the first try, the log line "Not scanned anymore. Access denied" is the second one. The third and fourth attempt is not logged in full by syslog, but instead as "last message repeated 2 times"), but it's actually scanned only once. The LRU mechanism denies access immediately for the second up to fourth attempt.

Figure 5.7 illustrates it **without** the LRU mechanism. The file is actually scanned four times - as showed also by the log snippet from F-Prot Daemon (5.8).

## 5.3 Performance

	ELF files (m:s)	Office files (m:s)
retrieving (1)	1:43	1:25
AntiVir (2)	0:40	0:33
McAfee uvscan (3)	1:28	1:04
KAV (4)	1:26	0:43
SAVSE (local) (5)	2:59	1:53
SAVSE ( $\rightarrow$ rlss2) (6)	4:08	1:51
WW (NAI) (local) (7)	3:57	3:38
WW (NAI) ( $\rightarrow$ rlss2) (8)	4:45	3:32
AvGuard (9)	2:31	1:55
kavmonitor (w/o cache) (10)	5:18	2:21
kavmonitor (w cache, 2500) (11)	5:19	1:33
kavmonitor (w cache, 7000) (12)	2:03	-
vscan-icap (SAVSE), local (13)	4:09	2:39
vscan-icap (SAVSE), $\rightarrow$ rlss2 (14)	5:26	3:03
vscan-icap (WW), local (15)	5:16	4:41
vscan-icap (WW), $\rightarrow$ rlss2 (16)	6:01	4:37
vscan-kavp (17)	4:46	2:20

Table 5.1: Samba-vscan: performance results

Performance tests have been done using Samba 2.2.8a, running on host rlss, and samba-vscan 0.3.3beta2. The ELF test included 6695 files (566 MB), the Office test 2220 files (517 MB) (same files as for 3.4.2, p. 40). The results of tests (2) up to (8) shows the elapsed time for the on-demand virus scan, i.e. without Samba integration at all.

- (1) time for retrieving files via "smbclient //localhost/test -N -Tc backup.tar" (done without any virus scanning, of course)
- (2) scan time using H+BEDV AntiVir / Linux Version 2.0.7 (on-demand)
- (3) scan time using NAI Virus Scan for Linux v4.24.0, Engine version 4.2.40, DAT version 4257 (on-demand), e.g. "uvscan /tmp/samba-test/office"
- (4) scan time using Kaspersky Anti-Virus for Linux, v4.0.3.0 (on-demand), e.g. "kavscanner /tmp/samba-test/office"
- (5) scan time using Symantec Anti Virus Scan Engine 4.0.3.41 (on-demand), e.g. "find /home/samba-test/office -type f -exec icap-client {} \;
- (6) same as (5), but SAVSE running on rlss2
- (7) scan time using WebWasher CSM 4.3 fcs, build 474, NAI Scan Engine version 4.2.40, DAT version 4257, on-demand, e.g. "find /home/samba-test/office -type f -exec icap-client {} \;
- (8) same as (7), but WW CSM running on rlss2
- (9) same as (1), but with kernel based on-access service AvGuard (Linux Version 2.0.7), Dazuko 1.1.2
- (10) same as (1), but with kernel based on-access service kavmonitor (KAV

4.0.3.0, kernel module v1.7, kavmonitor 1.9, kavdaemon 4.0.3.0), file cache **disabled** (CacheSize 0 in monitor.conf)

(11) same as (10), but with **enabled** file cache (with default CacheSize 2500)

(12) same as (11), but extended file cache (CacheSize 7500)

(13) same as (1), using the Samba VFS module vscan-icap (v0.3.3beta2), Symantec AntiVirus Engine (see (5))

(14) same as (13), SAVSE running on rlss2 (see (6))

(15) same as (1), using the Samba VFS module vscan-icap (v0.3.3beta2), WebWasher CSM (see (7))

(16) same as (15), WW CSM running on rlss2 (see (8))

(17) same as (1), using the Samba VFS module vscan-kavp (v0.3.3beta2), KAV (kavdaemon) 4.0.3.0

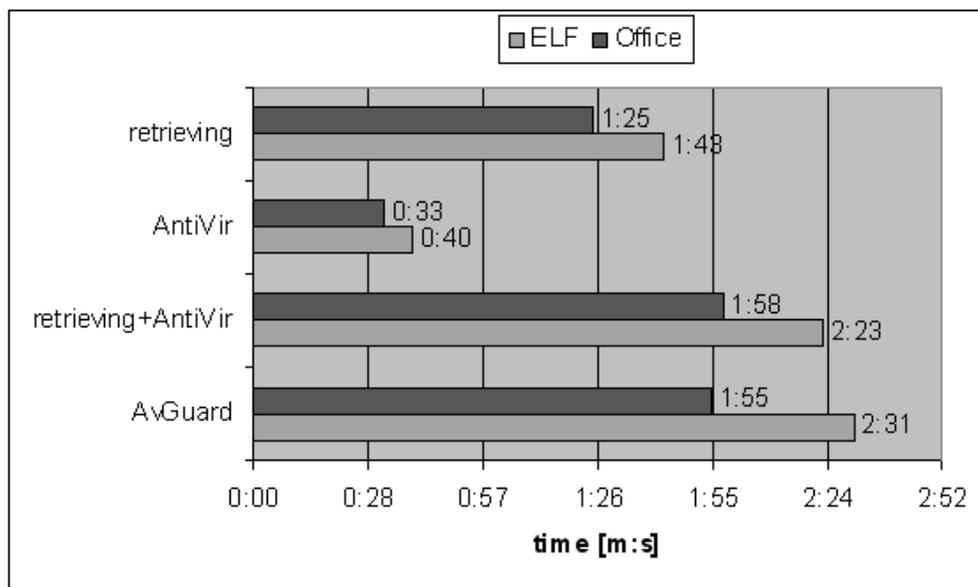


Figure 5.9: Performance results AvGuard + Samba

Figure 5.9 shows the results for H+BEDV AvGuard. The performance impact due to the kernel module Dazuko seems to be pretty small as the timing results of AvGuard are nearly the same as the summed up timing result for retrieving files via smbclient and the timing result of the on-demand test done with the command line scanner AntiVir. AvGuard currently misses a file cache, to cache results of previous scanned files. This would speed up AvGuard; and would avoid multiple scans of a file caused by Windows itself (as discussed in 5.2, p. 58).

By figure 5.10 the results for KAV are illustrated. For the Office file test, the results of vscan-kavp and kavmonitor (file cache disabled) are basically the same. And the elapsed time for both differs not that much from the summed up time for retrieving files (without any virus protection) and the scan time for the on-demand tests performed with KAV (kavscanner). For the ELF file

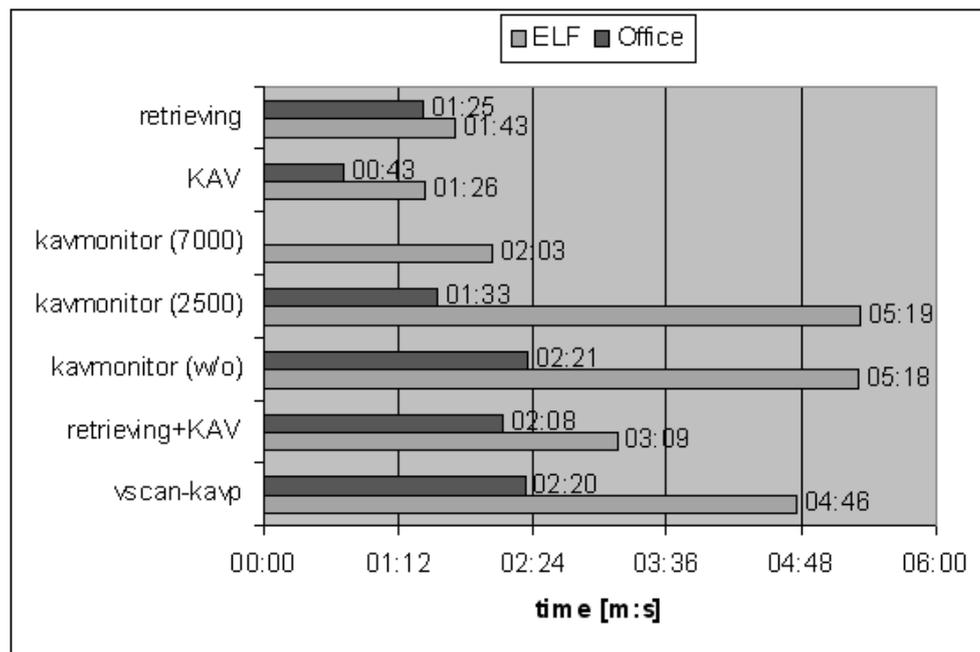


Figure 5.10: Performance results Samba, kavmonitor and vscan-kavp

test, the elapsed time result for vscan-kavp differs more than a minute from the summed up result (retrieving+KAV). The file cache of kavmonitor seems to be only efficient, when the maximum number of cache entries is at least equal to the number of files being scanned: only with max 7000 cache entries, and therefore more entries than files to be scanned in the ELF file test, a speed-up is noticeable.

The results for the Symantec Anti Virus Engine are showed in figure 5.11. Tests have been done with SAVSE running on localhost (L) and remote (R) on host rlss2. Those both tests were performed via "find ... type -f -exec ..." using the icap-client program, which means the icap-client utility has been started for each file. That's the reason why the summed up time retrieving and SAVSE is greater than the elapsed time for vscan-icap using SAVSE (the vscan-icap module is loaded only once when connecting to the Samba share, and not for every file being accessed). The difference between the time results "retrieving+SAVSE" and "vscan-icap (SAVSE)" varies.

The remarks for SAVSE are valid for the results of WebWasher with the NAI engine, too. NAI AV refers to the on-demand virus scanner "uvscan", which has been instructed to scan the corresponding directories, including sub-directories (e.g. "uvscan -recursive /tmp/samba-test/office")<sup>8</sup>. The difference between the time results "retrieving+WW" and "vscan-icap (WW)" is about 20 seconds (except for the ELF tests, remote).

<sup>8</sup>a timing test as "find /tmp/samba-test/office -type f -exec uvscan {} \;" took more than 20 minutes!

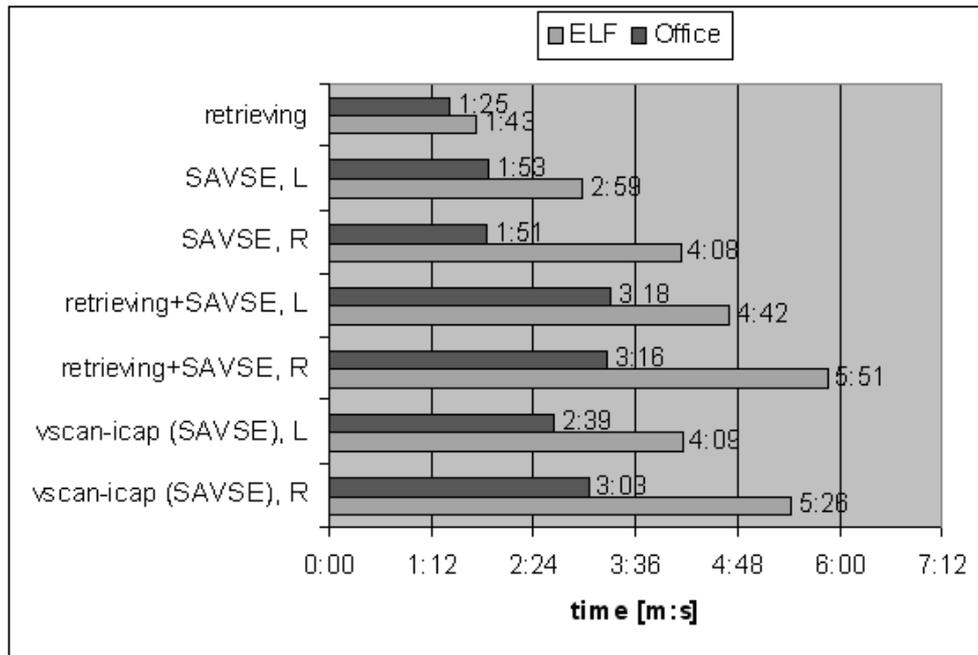


Figure 5.11: Performance results Samba, vscan-icap with SAVSE

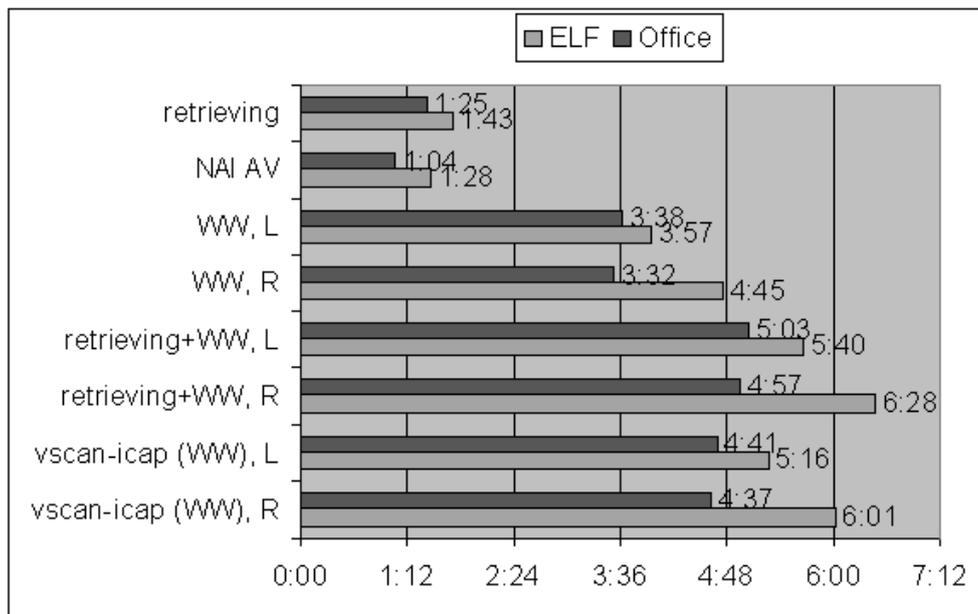


Figure 5.12: Performance results Samba, vscan-icap with WW CSM

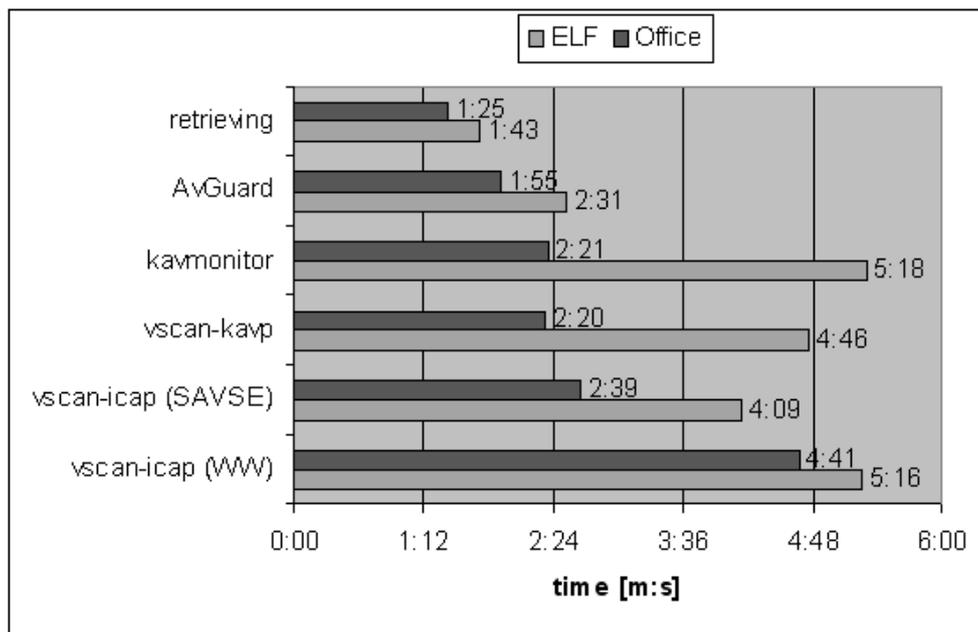


Figure 5.13: Performance results Samba (summary)

A summary of the Samba test results is shown by figure 5.13: the performance impact can be very huge! Therefore, future version of samba-vscan must take corresponding measures to avoid scan process as much as possible (like skipping certain file types; checksumming, to scan files with changed checksum only; file cache, so only not yet scanned or modified files are actually scanned). Whether a kernel-based solution or a Samba VFS-based solution is faster (i.e. lower overhead) can not be answered, as only Kaspersky can be used via both methods.

## 5.4 Conclusion

samba-vscan has been tested with Samba 2.2.x and Samba 3.0 up to alpha 21<sup>9</sup>. It provides the basic functionality, like denying access, sending notification messages and moving files into quarantine. Guessing from the feedback I received, the current set of features is sufficient for most people. The biggest problem as for now is speed - the performance loss caused by on-access scanning can be rather big. As mentioned, future versions must have an extended last recently used mechanism (file cache) to avoid the need of scanning file(s). The icap client code is currently just a proof-of-concept, as it used blocking I/O and ICAP's Preview method is not implemented. So, samba-vscan plus ICAP is not ready for production yet.

<sup>9</sup>any later version introduced changes to the VFS interface, which aren't fully supported by samba-vscan

Nevertheless, samba-vscan is used successfully in some smaller workgroup environments<sup>10</sup> - at least to my knowledge. Moreover, the software received positive feedback by commercial anti-virus companies<sup>11</sup>.

---

<sup>10</sup>see e.g. <http://computerdienst.kleinau.org/modules/tutorials/index.php?op=viewtutorial&tid=8>

<sup>11</sup>e.g. by Fridrik Skulason (Frisk Software Int.), <http://marc.theaimsgroup.com/?l=openantivirus-discuss&m=103580628204941&w=2> or by Paul Ducklin (Sophos Plc), <http://www.zdnet.com.au/newstech/security/story/0,2000048600,20274520,00.htm>

## Chapter 6

# FTP-/Web-Transfer - squid-icap

In this chapter, some concepts for protection FTP/Web transfer will be discussed. The most common used web-proxy on Linux/Unix is Squid, therefore we will focus on a solution for this proxy, using the ICAP protocol, named squid-icap. It will be compared with another proof-of-concept implementation, the squid-vscan project. As the Apache web-server can be configured to act as a proxy server, the mod-vscan project will be outlined as well.

### 6.1 Concepts

#### 6.1.1 Apache as Proxy

The web server Apache<sup>1</sup> can act as a web proxy by using the mod\_proxy module (and e.g. mod\_proxy\_http). The data, sent or received by the Apache server, can be processed by filters, which can be provided by a module ([AHD2003b]). The mod\_vscan project<sup>2</sup> by Kazutoshi Kubota implements a filter for virus scanning using ScannerDaemon of OpenAntiVirus.org. After building the module, the module can be used via the following changes to the Apache configuration file httpd.conf

```
<IfModule mod_vscan.c>
    SetOutputFilter VSCAN
</IfModule>
```

mod\_vscan allows some run-time configuration settings, like the path to the error document or to exclude some file-types from scanning ([KK2002]).

#### 6.1.2 Squid as Proxy

**squid-vscan**<sup>3</sup>, is a project by Kurt Huwig. It requires squid-filter by Olaf Titz, which implements filter capabilities for Squid<sup>4</sup>. A filter can be either a

---

<sup>1</sup>Apache 2.x is discussed here, [httpd.apache.org](http://httpd.apache.org)

<sup>2</sup>[http://www.willbe6.org/security/mod\\_vscan/](http://www.willbe6.org/security/mod_vscan/)

<sup>3</sup><http://www.openantivirus.org/projects.php>

<sup>4</sup><http://sites.inka.de/~bigred/devel/squid-filter.html>

request filter (working on the URI), a header filter (operating on the request or reply headers) or a content filter (operating on the reply body ([OT2002])). So, basically, squid-vscan is a new filter module, which works in conjunction with ScannerDaemon. By using squid-vscan, partial gets (i.e. retrieve only parts of a file, e.g. resuming a file transfer, which has not been completed before) are not possible anymore. This behaviour is needed to be able to scan the complete file. The "Filter" method ([KH2002]) is used for virus scanning, i.e. squid-vscan sends the "FILTER" command to ScannerDaemon, which responds with "Send/receive data to/from port '34709/34710' within 10 seconds" (the ports are chosen randomly). Then, the data will be scanned "on-the-fly": squid (squid-vscan) passes the data, received from the web server, directly to ScannerDaemon, which does the virus scanning and sends the data back to Squid (squid-vscan). Actually, sending a virus notification back to the client is not yet really implemented. In the case of a virus infection, both ports will be simply closed. Apart from the RPM installation, there's nothing to configure.

**squid-icap**<sup>5</sup>, is a patch which adds ICAP client facility to Squid. squid-icap is developed by HP Labs and WebWasher AG.

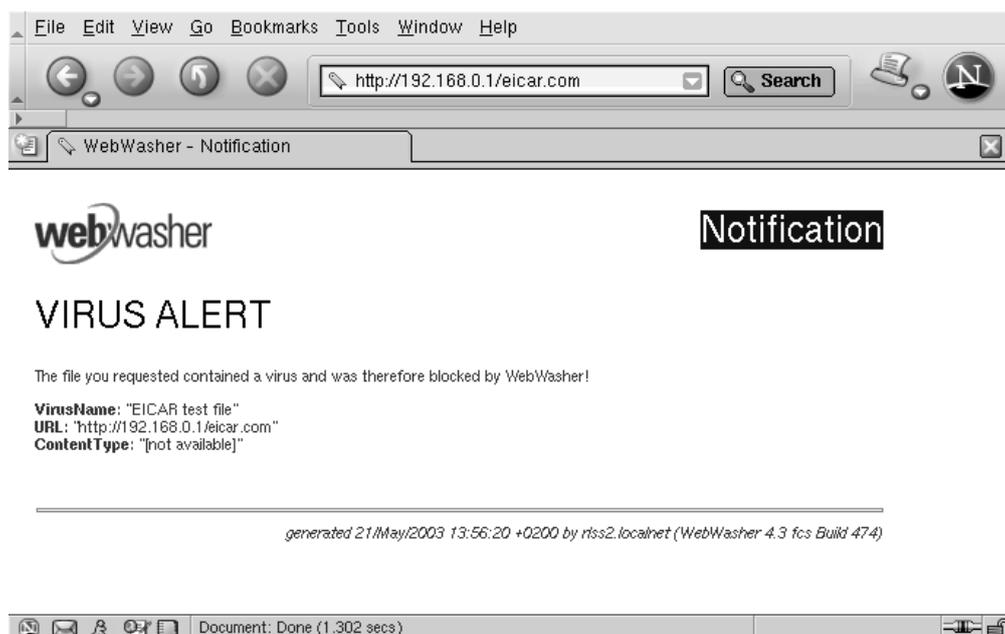


Figure 6.1: WebWasher virus notification page

<sup>5</sup><http://icap-server.sourceforge.net/squid.html>

squid-icap offers several directives in the squid configuration file squid.conf ([RH2002]). Settings used were

```
icap_enable on
icap_preview_size -1
icap_send_client_ip on
icap_service vscan respmod_precache 0 \
icap://127.0.0.1:1344/avscan
icap_class vscan_class vscan
icap_access vscan_class allow all
```

Note: the preview size will determined via the OPTION request (as well as other ICAP settings, like Transfer-Preview). The patched Squid will issue an OPTION request just after it has started and then after every five minutes.

Figure 6.1 shows the WebWasher virus notification page, displayed after an attempt to retrieve the eicar.com file.

## 6.2 Performance

Performance tests were done using three computers, as mentioned in section 3.4.1 (p. 39). The PII-266 MHz machine (IP: 192.168.0.1; NIC: 10 Mb/s) acts as Web server running Apache, the AMD Duron maschine (IP: 192.168.0.2; NIC: 100 Mb/s) runs Squid plus an anti-virus facility, the PIII-1GHz notebook (IP: 192.168.0.3; NIC: 100 Mb/s) played the Web client (using wget for fetching files). As anti-virus facility was used:

- WebWasher CSM 4.3 fcs using NAI Scan Engine (v4.2.40, DAT v4257)
- Symantec Anti Virus Scan Engine 4.0.3.41
- OpenAntiVirus ScannerDaemon 0.5.2

As proxy server

- Squid 2.5 STABLE1 including squid-icap 1.2.1<sup>6</sup>, used in conjunction with SAVSE and WW
- Squid 2.3 STABLE4 including squid-vscan 0.1.0<sup>7</sup>, used with ScannerDaemon

As for each request three connections are needed (client ↔ Squid, Squid ↔ Web server, Squid ↔ ICAP server) and due the 100 Mb/s network connection Squid is "flooded" with requests, the maximum number of open file descriptors must be increased, i.e. on a Linux system "ulimit -n 8192". I also increased the overall file descriptor limit, i.e. "echo "32768"> /proc/sys/fs/file-max"

<sup>6</sup>available at [http://sourceforge.net/project/showfiles.php?group\\_id=47737&release\\_id=119622](http://sourceforge.net/project/showfiles.php?group_id=47737&release_id=119622)

<sup>7</sup>available at [http://sourceforge.net/project/showfiles.php?group\\_id=10590&release\\_id=68273](http://sourceforge.net/project/showfiles.php?group_id=10590&release_id=68273)

	HTML files(m:s)	Image files (m:s)
Squid	0:27	0:30
Squid + SAVSE	4:14	0:37
Squid + WW CSM	3:13	0:37
WW CSM	0:31	0:31
squid-vscan	0:33	0:30

Table 6.1: Squid: performance results

Three tests have been performed:

- HTML files test: retrieving 6,631,430 Bytes in 1699 files, except for three GIF files all HTML files. Files retrieved via wget -m (mirroring, means retrieve all files)
- Image files test: 32,905,758 Bytes in 80 files, only very few HTML files, mostly JPG/GIF images. Again, fetched via wget -m
- retrieving three large GIF files (31MB, 72 MB and 144 MB) to measure the performance improvement by ICAPs Preview method

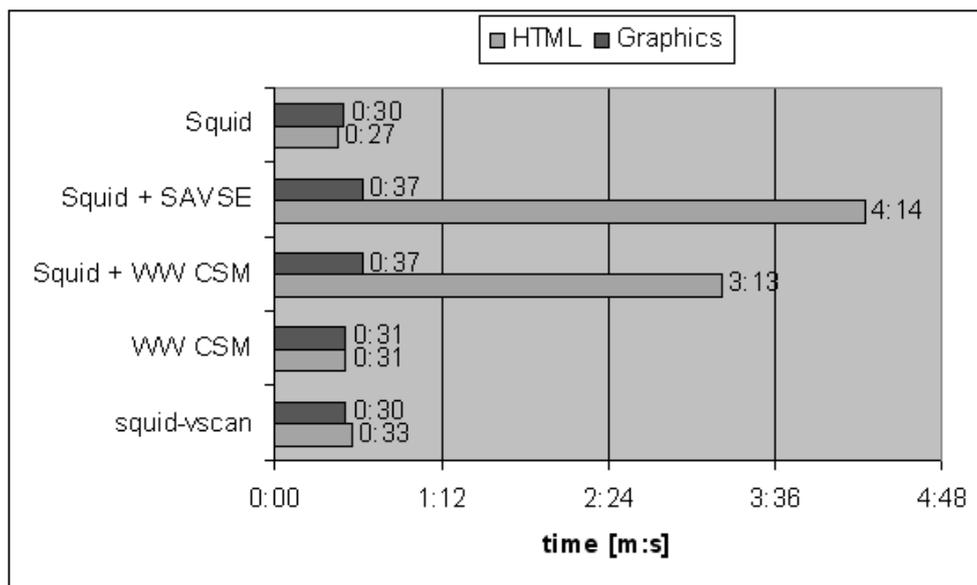


Figure 6.2: Performance results for Squid

Figure 6.2 shows the results of first two tests. It seems either the squid-icap code and/or WW CSM (acting as ICAP server) and SAVSE can not handle those numerous requests. As already mentioned, the proxy is "flooded" with HTTP GET requests issued by wget. Esp. when used with SAVSE, wget shows lots of "Connection closed at bye <bytepos>. Retrying.". At the

	31 MB (m:s)	72 MB (m:s)	144 MB (m:s)
Squid only	0:30	1:07	2:17
WebWasher	0:29	1:09	2:10
Squid+WW (no preview)	0:42	2:24	6:41
Squid+WW (preview)	0:41	2:05	5:39

Table 6.2: Squid: performance results retrieving large GIF file

same time "tail -f cache.log" (one of Squid logs file) showed "comm\_write: fd\_table[<num>].rwstate != NULL". Therefore, it took a long time to retrieve all HTML files. The results for the image file test are better, because it contains only 80 files and by the large files the proxy is not flooded with requests within a second (as downloading takes some seconds).

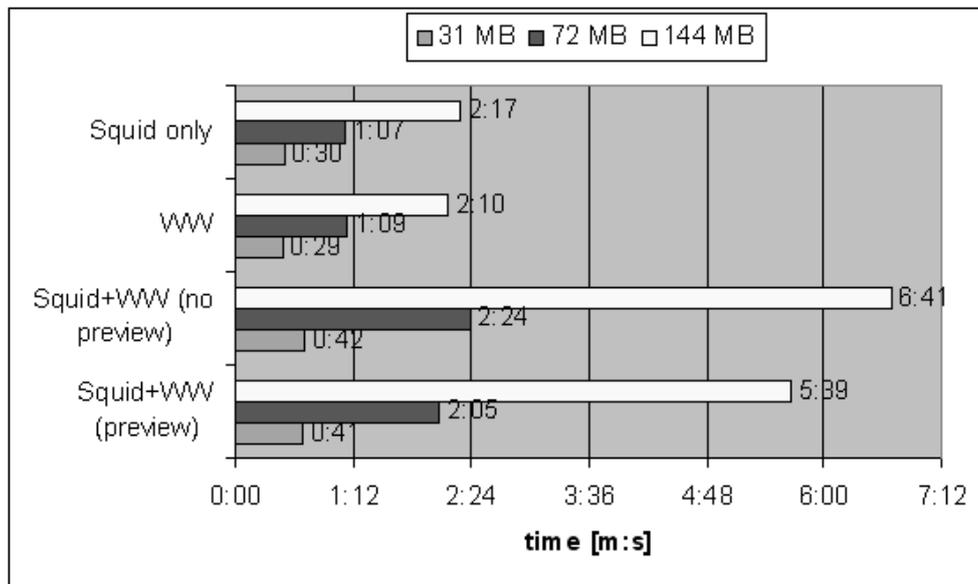


Figure 6.3: Performance results for Squid retrieving large GIF file

The last test was retrieving some larger GIF files to measure the performance loss, when Preview was disabled. With ICAP's preview mechanism, only 30 bytes are actually transferred to the ICAP server (here WebWasher). Without preview, the complete file must be transferred to the ICAP server which takes some time, obviously. By retrieving the 144 MB GIF file, the Preview method saved about one minute. But even using this method, the download took very long. Actually, squid-icap retrieves the complete file first, before sending it to the client (a method which I would call "store-and-forward"). Of course, this behaviour is required, when retrieving e.g. an EXE or an ZIP file (here, the complete data must be sent to the ICAP server for scanning first). But in the GIF file case, the data must not be sent to the ICAP server (which has, in fact,

send an "204 No modification needed") - so, the data can be "piped-through" to the client.

Here are the corresponding tcpdump log snippets (modified/simplified) to illustrate how squid-icap client works:

First step, 192.168.0.3 (client) sends an HTTP GET request to 192.168.0.2 (proxy)

```
14:37:22.862026 192.168.0.3.34883 > 192.168.0.2.3128: S
14:37:22.862131 192.168.0.2.3128 > 192.168.0.3.34883: S
14:37:22.862284 192.168.0.3.34883 > 192.168.0.2.3128: . ack
14:37:22.863062 192.168.0.3.34883 > 192.168.0.2.3128: P
14:37:22.863157 192.168.0.2.3128 > 192.168.0.3.34883: . ack
```

Squid starts to retrieve the requested file from the Web server (192.168.0.1)

```
14:37:22.863836 192.168.0.2.36473 > 192.168.0.1.80: S
14:37:22.864511 192.168.0.1.80 > 192.168.0.2.36473: S
14:37:22.864613 192.168.0.2.36473 > 192.168.0.1.80: . ack
14:37:22.864998 192.168.0.2.36473 > 192.168.0.1.80: P
```

Squid sends an ICAP preview request to the ICAP server (listening on 127.0.0.1), shown here in full-length (i.e. not shortened)

```
14:37:22.865195 127.0.0.1.36474 > 127.0.0.1.1344: S
14:37:22.865257 127.0.0.1.1344 > 127.0.0.1.36474: S
14:37:22.865295 127.0.0.1.36474 > 127.0.0.1.1344: . ack
14:37:22.895595 127.0.0.1.36474 > 127.0.0.1.1344: P
14:37:22.895664 127.0.0.1.1344 > 127.0.0.1.36474: . ack
14:37:22.896734 127.0.0.1.1344 > 127.0.0.1.36474: P
14:37:22.896784 127.0.0.1.36474 > 127.0.0.1.1344: . ack
14:37:22.896906 127.0.0.1.1344 > 127.0.0.1.36474: F
14:37:22.897347 127.0.0.1.36474 > 127.0.0.1.1344: F
14:37:22.897383 127.0.0.1.1344 > 127.0.0.1.36474: .
```

Squid is still retrieving the file, the download is finished nearly two minutes later

```
14:39:39.836020 192.168.0.1.80 > 192.168.0.2.36473: P
14:39:39.836055 192.168.0.2.36473 > 192.168.0.1.80: . ack
14:39:55.949254 192.168.0.1.80 > 192.168.0.2.36473: F
14:39:55.949431 192.168.0.2.36473 > 192.168.0.1.80: F
14:39:55.949888 192.168.0.1.80 > 192.168.0.2.36473: . ack
```

Now, Squid sends the data back to the client

```
14:39:39.836746 192.168.0.2.3128 > 192.168.0.3.34883: .
14:39:39.836789 192.168.0.2.3128 > 192.168.0.3.34883: .
14:39:39.836813 192.168.0.2.3128 > 192.168.0.3.34883: P
```

```
14:39:39.837316 192.168.0.3.34883 > 192.168.0.2.3128: . ack
14:39:39.837422 192.168.0.2.3128 > 192.168.0.3.34883: .
14:39:39.837440 192.168.0.2.3128 > 192.168.0.3.34883: .
14:39:39.837369 192.168.0.3.34883 > 192.168.0.2.3128: . ack
[. .]
14:43:45.707313 192.168.0.2.3128 > 192.168.0.3.34883: F
14:43:45.707694 192.168.0.3.34883 > 192.168.0.2.3128: F
14:43:45.707775 192.168.0.2.3128 > 192.168.0.3.34883: . ack
```

When retrieving a large file or e.g. a ZIP-file, which needs to be downloaded in full by the proxy and requires long time until virus scanning is finished, the browser may time-out, as no data is received. Or, the user may cancel the download, as he's under the impression of a slow Internet connection. Several workarounds exists for this problem, e.g. forwarding some bytes to the browser. But as this is a generic problem and not related to the ICAP protocol, it's not discussed here. See [MST2002] for details and a suggested solution.

### 6.3 Conclusion

Squid (including the squid-icap) worked flawlessly in my tests. I used it with SAVSE for normal, every day browsing and the performance loss was very low. Nevertheless, it should be considered to improve performance of files which need not to be scanned by not "buffering" the complete file. Instead, forward the data immediately to the client in this particular case.



## Chapter 7

# Résumé

As shown on the last three chapters, server-based virus protection running on Unix/Linux servers (tier 2 and 3) is possible. As the mentioned tools support the ICAP protocol, an ICAP anti-virus service (running on a dedicated host) may serve as multi-purpose virus scanning facility for mail, web and proxy server(s). By this concept the resource-intensive virus scanning task can be off-loaded. This might be interesting esp. for CD-only based firewalls or appliances with limited (hardware) resources. Or, if the preferred anti-virus vendor does not offer a version for the OS (say AIX) the mail/file/proxy server runs, one might add a Linux box running the ICAP enabled anti-virus server. Migrating from anti-virus vendor A to B should be in most cases very easy; but the choice is currently very limited as only very few products supporting ICAP are available.

AMaViS, the oldest and therefore the most stable and mature solution discussed in this thesis, serves low- and medium-volume mail servers. Squid (squid-icap) worked stable and is worth a look. samba-vscan is most likely only usable for smaller workgroups at the moment.

ICAP will be most likely superseded by OPES sooner or later. OPES looks promising, although it's not specified in details yet. Currently, it can not be foreseen when a "stable" draft status is reached which allows a first implementation.

Of course, I will continue to work on AMaViS and samba-vscan. For the latter one better ICAP support (i.e. ICAP Preview method), exclusion of files based on file type and improved file cache are planned. And I will follow the ICAP/OPES discussion(s) on the relevant mailing lists.



# List of Figures

1.1	Payload of Ambulance Car virus . . . . .	2
1.2	Overwriting / Appending virus . . . . .	3
1.3	Screenshot of VBS virus/worm construction kit . . . . .	8
2.1	Connection invoking a Content Vectoring Server . . . . .	21
2.2	CVP data flows . . . . .	22
3.1	Request Modification . . . . .	24
3.2	Request Satisfaction . . . . .	24
3.3	Response Modification . . . . .	25
3.4	Result Modification . . . . .	26
3.5	Symantec AntiVirus Scan Engine . . . . .	27
3.6	Web Washer CSM suite . . . . .	29
3.7	Performance results AVI file test, WebWasher, host rlss . . . . .	42
3.8	Performance results AVI file test, WebWasher, host rlss2 . . . . .	42
3.9	Performance results AVI file test, SAVSE, host rlss . . . . .	43
3.10	Performance results AVI file test, SAVSE, host rlss2 . . . . .	43
3.11	Performance results ELF/Office file test, SAVSE . . . . .	44
3.12	Performance results ELF/Office file test, WebWasher . . . . .	45
3.13	Performance results ELF/Office file test, NAI AV, WebWasher . . . . .	45
4.1	Sendmail: filter example . . . . .	50
4.2	Postfix: simple content filtering example . . . . .	51
4.3	Postfix: advanced content filtering example . . . . .	52
5.1	Samba VFS (simplified) . . . . .	57
5.2	Samba-vscan: access denied . . . . .	59
5.3	Samba-vscan: virus notification . . . . .	59
5.4	Samba-vscan: architecture . . . . .	61
5.5	Samba-vscan: single-click on PowerPoint file . . . . .	62
5.6	Samba-vscan: double click on infected Word file I . . . . .	63
5.7	Samba-vscan: double click on infected Word file II . . . . .	63
5.8	F-ProtD: double click on infected Word file . . . . .	64
5.9	Performance results AvGuard + Samba . . . . .	66
5.10	Performance results Samba, kavmonitor and vscan-kavp . . . . .	67
5.11	Performance results Samba, vscan-icap with SAVSE . . . . .	68
5.12	Performance results Samba, vscan-icap with WW CSM . . . . .	68

5.13	Performance results Samba (summary) . . . . .	69
6.1	WebWasher virus notification page . . . . .	72
6.2	Performance results for Squid . . . . .	74
6.3	Performance results for Squid retrieving large GIF file . . . . .	75

# List of Tables

3.1	Service Architecture Summary . . . . .	26
3.2	Available ICAP AntiVirus Servers . . . . .	27
3.3	Performance results for AVI file test . . . . .	41
3.4	Performance Results for ELF/Office files test . . . . .	44
4.1	Milter callbacks related to SMTP transaction . . . . .	50
5.1	Samba-vscan: performance results . . . . .	65
6.1	Squid: performance results . . . . .	74
6.2	Squid: performance results retrieving large GIF file . . . . .	75



# Bibliography

- [FC1984] Fred Cohen. *Computer Viruses - Theory and Experiments*. Chapter 2 - A Computer virus. <http://www.all.net/books/virus/part2.html>.
- [BSI1994] BSI. *Informationen zu Computer-Viren*. Schriftenreihe zur IT-Sicherheit, Band 2, Juni 1994.
- [CS1995] CHIP Special. *Computer-Viren '95*. Vogel Verlag, 1995.
- [MR1995] Martin Roesler. *FAQ der VIRUS.GER - Version 2.3*. <http://www.vhm.haitec.de/faq/>
- [FSC2003] F-Secure Corp. *F-Secure Virus Screenshots Archive*. F-Secure Corporation, 2003. <http://www.f-secure.com/virus-info/v-pics/>
- [AM2000] Andreas Marx. *Die Geschichte der Computerviren*. 1997/2000. <http://www.vhm.haitec.de/konferenz/1997/history.htm>
- [FP2000] François Padget. *Computer Viruses: The Technological Leap*. EICAR Conference 2000, [http://download.nai.com/products/media/vil/rtf/fpaget\\_EICAR\\_2000.rtf](http://download.nai.com/products/media/vil/rtf/fpaget_EICAR_2000.rtf)
- [EK2001] Eugene Kaspersky and Andy Nikishin. *Back to the future - again*. Proceedins of Virus Bulletin Conference 2001.
- [RS1996] Rune Skardhamar. *Virus Detection And Elimination*, 1996. Academic Press, Inc.
- [VBM1998] *What the HTML?*. Virus Bulletin Magazine, December 1998, Virus Bulletin Ltd.
- [CR1999] Costin Raiu. *A Hill of Beans*. Virus Bulletin Magazine, March 1999, Virus Bulletin Ltd.
- [PS1999] Peter Szor. *Beast Regards*. Virus Bulletin Magazine, June 1999, Virus Bulletin Ltd.
- [ANMK1999] Andi Nikishin & Mike Pavluschick. *pOLEmorphism*. Virus Bulletin Magazine, June 1999, Virus Bulletin Ltd.

- [NF1999] Nick FitzGerald. *Gala Premiere*. Virus Bulletin Magazine, July 1999, Virus Bulletin Ltd.
- [IM1999a] Dr Igor Muttik. *Macro Viruses - Part 1*. Virus Bulletin Magazine, September 1999, Virus Bulletin Ltd.
- [IM1999b] Dr Igor Muttik. *Macro Viruses - Part 2*. Virus Bulletin Magazine, October 1999, Virus Bulletin Ltd.
- [VB1996] Vesselin Bontchev. *Possible Macrovirus Attacks and How to Prevent Them*. Proceedings of Virus Bulletin Conference 1996.
- [SB1996] Steve Baily. *Unix viruses - a feasibility study*. Proceedings of the Virus Bulletin Conference 1996.
- [DAJ1997] Dr. David Aubrey-Jones. *Macro Attacks On Office97*. Proceedings of Virus Bulletin Conference 1997.
- [MO1997] Martin Overton. *FAT32 - new problems for anti-virus, or viruses?*. Proceedings of Virus Bulletin Conference 1997.
- [VB1997] Vesselin Bontchev. *Macrovirus identification problems*. Proceedings of Virus Bulletin Conference 1997.
- [VBKT2002] Vesselin Bontchev & Katrin Tocheva. *Macro and script virus polymorphism*. Proceedings of Virus Bulletin Conference 2002.
- [SK1999] Stefan Kurtzhals. *Aktuelle Entwicklungen im Bereich der Makroviren*. VIRUS.GER-Conference 1999. <http://www.vhm.haitec.de/konferenz/1999/makroviren/>
- [JK1998] Jakub Kaminski. *Disappearing Macros - Natural Devolution of Upconverted Macros*. Proceedings of Virus Bulletin Conference 1998.
- [VB1998] Vesselin Bontchev. *The 'Pros' and 'Cons' of WordBasic Virus Upconversion*. Proceedings of Virus Bulletin Conference 1998.
- [MH1998] Mikko Hyponnen. *Putting Macros under Control*. Proceedings of Virus Bulletin Conference 1998.
- [PS1998] Peter Szor. *Attacks on Win32*. Proceedings of Virus Bulletin Conference 1998.
- [RJZ1999] Righard J Zwienenberg. *Office Millenium: what does it (not) give?*. Proceedings of Virus Bulletin Conference 1999.
- [KT1999] Katrin Tocheva. *Multiple Infections*. Proceedings of Virus Bulletin Conference 1999.
- [MVO1999] Martin van Oers. *Automatic MS Outlook VB Script*. Proceedings of Virus Bulletin Conference 1999.

- [JK1999] Jakub Kaminiski. *Hide and Seek - Stealth Techniques of Macro Viruses*. Proceedings of Virus Bulletin Conference 1999.
- [CN1999] Carey Nachenberg. *Computer Parasitology*. Proceedings of Virus Bulletin Conference 1999.
- [CN1996] Carey Nachenberg. *Understanding and Managing Polymorphic Viruses*. The Symantec Enterprise Papers. Symantec, 1996.
- [TM2002] Taras Malivanchuk. *The Win32 Worms: classification and possibility of heuristic detection*. Proceedings of Virus Bulletin Conference 2002.
- [PS2000] Peter Szor. *Attacks on Win32 - Part II*. Proceedings of Virus Bulletin Conference 2000.
- [JK2000] Jakub Kaminski. *Linux Malware - has the next battlefield been decided?*. Proceedings of Virus Bulletin Conference 2000.
- [MVO2000] Marius van Oers. *Linux Viruses - ELF file format*. Proceedings of Virus Bulletin Conference 2000.
- [JK2001] Jakub Kaminski. *Not so quiet on the Linux front: Linux Malware II*. Proceedings of Virus Bulletin Conference 2001.
- [ECPZ2002] Eric Chien & Peter Szor. *Blended Attacks. Exploits, vulnerabilities and buffer-overflow techniques in computer viruses*. Proceedings of Virus Bulletin Conference 2002.
- [KC2002] Katherine Carr. *Sophos Anti-Virus detection: a technical overview*, Sophos, Oxford, UK, October 2002, <http://www.sophos.com/sophos/docs/eng/papers/sav-overview.pdf>
- [AM2002] av-test.org, GEGA IT GbR, Andreas Marx *Comparison Test 2002-01*, category "archived and compressed file formats" <http://www.av-test.org/online/sites/os03.php3?js=0&test=2002-01&p=1>
- [CGR2002] Costin G. Raiu. *OpenAV: Developing Open Source AntiVirus Engines*. <http://www.securityfocus.com/infocus/1650>
- [IM2000] Igor Muttik. *Stripping down an AV engine*. Proceedings of Virus Bulletin Conference 2000.
- [FF2001] Francisco Fernandez. *Heuristic Engines*. Proceedings of Virus Bulletin Conference 2001.
- [MS2002] Markus Schmall. *Heuristic Techniques in AV Solutions: An Overview*. <http://www.securityfocus.com/infocus/1542>
- [RM2002] Richard Marko. *Heuristics: Retroperpective and future*. Proceedings of Virus Bulletin Conference 2002.

- [CN2002] Carey Nachenberg. *Behavior Blocking: The Next Step in Anti-Virus Protection*. <http://www.securityfocus.com/infocus/1557>
- [LL2001] Lixim Lu. *Reducing false positives in behaviour blocking*. Proceedings of the Virus Bulletin Conference 2001.
- [AN1999] Andy Nikishin. *Advantages and Disadvantages of modern integrity checkers*. Proceedings of Virus Bulletin Conference 1999.
- [ICSA2002] ICSA Labs, Larry Bridwell. *ICSA Labs 8th Annual Computer Virus Prevalence Survey 2002*. ICSA Labs, a division of TruSecure Corp. <http://www.trusecure.com> (registration needed)
- [SAVI] Sophos Plc. *SAVI Developer Toolkit - User manual*. <http://www.sophos.com/support/docs/#savi>
- [WRS1992] W. Richard Stevens. *Advanced Programming in the UNIX® Environment*. Addison-Wesley, 1992. Seventh Printing, September 1994.
- [LH2001] Lars Hecking. *Sophie experiences*. Posting to the amavis-user mailing list, 08/15/2001. <http://marc.theaimsgroup.com/?l=amavis-user&m=99788946018453&w=2>
- [RFC1945] T. Berners-Lee et al. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. May, 1999. Obsoleted by RFC 2068/2616. <ftp://ftp.rfc-editor.org/in-notes/rfc1945.txt>
- [CVP2002] Check Point Software Technologies Ltd. *Check Point™ VPN-1/FireWall-1® CVP (Content Vectoring Protocol) API Specification*, NG FP3. September 2002. Available at <http://www.opsec.com> after a registration has been filed.
- [KA2001] Konstantin Agouros *How to write a CVP-Client?*. Posting to newsgroup cp.products.opsec.sdk, 5 Oct 2001 21:58:25 +0200, Message-ID: <elwood.1002311864@news.agouros.de>.
- [ICAP01] Network Appliance. *Internet Content Adaptation Protocol (ICAP)*. Version 1.01, 7/30/01. [http://www.i-cap.org/docs/icap\\_whitepaper\\_v1-01.pdf](http://www.i-cap.org/docs/icap_whitepaper_v1-01.pdf)
- [RFC3507] J. Eslon et al. *Internet Content Adaptation Protocol (ICAP)*. RFC 3507. April 2003. <http://www.ietf.org/rfc/rfc3507.txt>
- [RFC2616] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. <ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt>
- [MST2003] Martin Stecher. *ICAP Extensions*. Draft. April, 2003. <http://www.martin-stecher.de/draft-stecher-icap-subid-00.txt>

- [FB2002] Frank Berzau. *Summery of the ICAP Discussion*. Presentation for the 55th IETF Meeting in Atlanta, Georgia, November 17-21, 2002. <http://www.ietf.org/proceedings/02nov/slides/opes-3.pdf>
- [WRS2000] W. Richard Stevens. *Programmieren von Unix Netzwerken. Netzwerk APIs: Sockets und XTI*. Carl Hanser Verlag, 2000, 2. Auflage.
- [SAVSE02] Symantec Corporation. *Symantec AntiVirus<sup>TM</sup> Scan Engine*. Documentation version 4.0, 12/2002. Symantec Corporation, 2000-202 (part of the SAVSE product).
- [SSS02] Symantec Corporation. *Symantec AntiVirus<sup>TM</sup> Scan Engine Software Developer's Guide*. Documentation version 4.0. Symantec Corporation, 2000-2002.
- [OPS03] IETF OPES WG. *Open Pluggable Edge Services (opes)*. <http://www.ietf.org/html.charters/opes-charter.html>
- [ABA2002] A. Barbir et al. *OPES Use Cases and Deployment Scenarios*. Internet Draft. Aug 5, 2002. Expires: February 5, 2002. <http://www.ietf.org/internet-drafts/draft-ietf-opes-scenarios-01.txt>
- [AR2003] A. Rousskov. *OPES Callout Protocol Core*. Internet Draft. April 10, 2003. Expires: October 9, 2003. <http://www.measurement-factory.com/tmp/opes/snapshots/latest/ocp-spec.html>
- [RL2003] Rainer Link et al. *AMaViS & sendmail*. README.sendmail, part of the AMaViS documentation, <http://cvsweb.amavis.org/amavis/README.sendmail>
- [GW2003] Geoff Winkless et al. *How to use AMaViS with sendmail/libmilter*. README.milter, part of the AMaViS documentation, <http://cvsweb.amavis.org/amavis/README.milter>
- [SI2000] Sendmail Inc. *Filtering Mail with Sendmail*. [http://www.sendmail.com/partner/resources/development/milter\\_api/index.html](http://www.sendmail.com/partner/resources/development/milter_api/index.html)
- [WV2003] Wietse Venema. *Postfix Content Filtering*. FILTER\_README, part of the postfix distribution, <http://www.postfix.org>
- [LH2003] Lars Hecking et al. *How To Use AMaViS With postfix*. README.postfix, part of the AMaViS documentation, <http://cvsweb.amavis.org/amavis/README.postfix>
- [CH2001] Chris Hertel. *Samba: An Introduction*. <http://de.samba.org/samba/docs/SambaIntro.html>

- [RS2002] Richard Sharpe. *Just what is SMB?*. <http://www.samba.org/cifs/docs/what-is-smb.html>
- [CRH2003] Christopher R. Hertel. *Implementing CIFS*. <http://www.ubiqx.org/cifs/Intro.html#INTRO.2>
- [DBMC2001] Daniel P. Bovet & Marco Cesati. *Understanding the Linux kernel*. O'Reilly & Associates, Inc., 2001.
- [JO2003] John Ogness, H+BEDV Datentechnik GmbH. *Dazuko project*, <http://www.dazuko.org/>
- [DCB2000] Dave Collier-Brown. *VFS tutorial*. [http://www.geocities.com/orville\\_torpid/papers/vfs\\_tutorial.html](http://www.geocities.com/orville_torpid/papers/vfs_tutorial.html) (outdated)
- [AB2002] Alexander Bokovoy. *Extending Samba with cascaded VFS modules*. Proceedings of SambaXP 2002 conference. [http://www.sambaxp.org/sambaXP\\_2002/archive.html](http://www.sambaxp.org/sambaXP_2002/archive.html)
- [MSC2002] Microsoft Corp. *Popup Message Queue Can Only Receive 6 Messages*. Microsoft Knowledge Base Article - 132524, <http://support.microsoft.com/default.aspx?scid=kb;en-us;132524>
- [MP2001] Martin Pool. *Re: VFS: connect to share is different with Win98 and smbclient*. Posting to samba-technical mailing list, <http://marc.theaimsgroup.com/?l=samba-technical&m=100632388426164&w=2>
- [AHD2003a] Apache HTTP Server Documentation Project. *Apache module mod\_proxy*. Apache HTTP Server 2.0 Documentation. [http://httpd.apache.org/docs-2.0/mod/mod\\_proxy.html](http://httpd.apache.org/docs-2.0/mod/mod_proxy.html)
- [AHD2003b] Apache HTTP Server Documentation Project. *Filters*. Apache HTTP Server 2.0 Documentation. <http://httpd.apache.org/docs-2.0/filter.html>
- [KK2002] Kazutoshi Kubota. *mod\_vscan*. [http://www.willbe6.org/security/mod\\_vscan/](http://www.willbe6.org/security/mod_vscan/)
- [OT2002] Olaf Titz. *Filter modules for Squid*. <http://sites.inka.de/~bigred/devel/squid-filter.html>
- [KH2002] Kurt Huwig. *ScannerDaemon Howto*. [http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/\\*checkout\\*/openantivirus/java/doc/ScannerDaemon-Howto.html?rev=1.9](http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/*checkout*/openantivirus/java/doc/ScannerDaemon-Howto.html?rev=1.9)
- [RH2002] Ralf Horstmann. *Squid ICAP Client Configuration*. <http://icap-server.sourceforge.net/icap-configuration.html>
- [MST2002] Martin Stecher. *LateClearance Content Encoding*. Internet Draft. October, 2002. Expires April, 2003. <http://www.ietf.org/internet-drafts/draft-stecher-lclr-encoding-00.txt>

# Appendix A

## GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License

principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of

pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the

copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give

---

them a chance to provide you with an updated version of the Document.

#### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and

list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on

covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number.

If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.