A STREAM-BASED MAIL PROXY WITH INTERLEAVED DECOMPRESSION AND VIRUS SCANNING

Ying-Dar Lin[1], Szu-Hao Chen[1], Po-Ching Lin[1] and Yuang-Chen Lai[2]
[1]Department of Computer Science
National Chiao Tung University, HsinChu, Taiwan
[2]Department of Information Management
National Taiwan University of Science and Technology, Taipei, Taiwan
Email: [1]{ydlin, cwjan, pclin}@cis.nctu.edu.tw, [2]laiyc@cs.ntust.edu.tw

ABSTRACT
Anti-virus systems can operate on access gateways for centralized management and early virus blocking. When serving a group of computers, the traditional storage-based mechanism is less scalable because the mail should be stored. This work designs a stream-based mail proxy that processes the mail segment by segment without storing the entire mail and interleaves the MIME parsing, decoding, decompression and virus scanning. We integrate several modified open source packages into the proxy and use the system call *select* to achieve single-process concurrency. The benchmarking reveals this new proxy is seven times faster than the storage-based one on simply forwarding, three times faster on virus scanning, and twice faster on both file decompression and virus scanning. This proxy keeps nearly constant memory consumption and works without disk storage, while the storage-based proxy requires the disk space to be proportional to the number of clients and the mail size.

KEY WORDS
viruses and worms, stream-based, mail proxy, decompression.

1. Introduction

Conventionally, anti-virus systems run on host computers. Since most infections come from outside networks, blocking viruses on the access gateway appears to be a trend. Such gateway-based centralized management could reduce the cost of maintaining the anti-virus system on a number of internal host computers. Virus scanning on the gateway can be storage-based and stream-based. The former receives the entire mail content before scanning, while the latter scans the part that has been received and sends it out immediately after the scanning. The storage-based scanning has poor scalability in storage. For example, if 10,000 connections send 500 KB files concurrently, the total storage occupying the gateway will be 5 GB. The system needs large storage and hence is costly.

By interleaving receiving, scanning and sending, the required memory buffer size in a connection can be ideally kept *constant* rather than *proportion to* the file size. All the components in the processing flow should be also stream-based. For instance, the mail content may be MIME encoded, compressed and encrypted. Fortunately, the decoding and decompression can be streamed-based, i.e. interleaved.

This work implements a stream-based mail proxy with interleaved decompressing and virus scanning. Several open source packages are selected to be integrated: Net::SMTP::Server [1] as the SMTP protocol handler and another modified version as the POP3 handler, ClamAV [2] for anti-virus, and Zlib [3] + Compress::Zlib [4] for file decompression. For better performance and lower memory consumption, the system is implemented as a single-process proxy. A series of external and internal benchmarks are performed after the integration. This proxy is compared with AMaViS [5] in terms of throughput, latency, and the space required in memory and disk. We intend to study the questions. (1) How can file decompression and virus scanning be interleaved seamlessly? (2) By how much can the stream-based proxy improve the scalability and performance? (3) How heavy are the decompression and virus scanning compared with other components?

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 describes the design issues. The system architecture and workflow are presented in Section 4. Section 5 details the system implementation. Section 6 presents the benchmark results of both the stream-based and the storage-based systems. Section 7 concludes this work.

2. Related Works

Most commercial products are storage-based, such as InterScan messaging Security Suite from TrendMicro [6], FortiGate series from Fortinet [7] and F-pod series from FRISK Software [8]. The open source project AMaViS is also storage-based. Until March 2005, the only commercial stream-based anti-virus gateway is Content Security Gateway from CPScure [9], but its inner working is unknown since it is a black box. The open source project Anomy [10] is a mail-sanitizing tool on F-pod. Its MIME parser treats mail as a stream of data, but the attachment is still processed in a storage-based fashion. One reason that storage-based anti-virus systems still dominate the market is they are versatile in handling an infected file, such as quarantine that stored the infected file for later retrieval. A stream-based anti-virus system simply drops the infected file, and so the file is destroyed.

The concept of stream-based design has been existent in other application domains in the research field. A cut-through switch can send out a portion of a packet before the entire packet is received. A "segment-based proxy cache of multimedia streams" [11] treats the whole video as variable-sized segments. Chi et al. [12] discussed on-the-fly compression/decompression on a Web proxy.

3. Design issues

3.1 Overheads in a storage-based mail proxy

AMaViS is selected for observing the overheads of a storage-based mail proxy because of its popularity. AMaViS is a Perl program and acts as an interface connecting two mail transport agents (MTAs). An MTA receives mail from port 25 and passes the mail to AMaViS for virus scanning. If no virus is found, AMaViS transmits this mail to another MTA that relays it to the target mail server. Three overheads are in the process: (1) file access, (2) inter-process communications and (3) process forking in AMaViS. For (1), AMaViS receives the mail and decodes attachments into files. If the files need to be decompressed, AMaViS calls an external program to decompress them into other files, and then calls the virus scanner to scan these files. For (2), inter-process communications exist between AMaViS and the two MTAs. They also occur when AMaViS calls the external programs for file decompression and virus scanning. For (3), a per-client process is forked for each incoming connection. These processes occupy the memory and the *fork* system call is heavy.

3.2 Overheads in a storage-based mail proxy

The essential requirement of a stream-based mail proxy is that each component in the proxy should be stream-based. The processing includes MIME parsing, decoding, file decompressing, virus scanning and encoding. The proxy receives a part of a mail in a memory buffer, and then processes the buffer according to its content. For example, decompressing and decoding require extra buffers. The processing is on the buffer rather than on the entire file.

We prefer a single-process proxy with socket I/O multiplexing to handle concurrency because the implementation of Perl threads is inefficient [14]. The single-process architecture is memory efficient and works without context-switching overheads. There is also no thread synchronization and inter-process communications. This architecture could render high scalability in terms of the number of connections.

A storage-based system stores the decompressed files, which may be much larger than the original files, making a denial-of-service attack possible. The storage-based system thus often bypasses or blocks the file whose size exceeds a threshold after the decompression. Fortunately, stream-based file decompression is feasible for most compression formats according to our survey. Table 1 summarizes the feasibility of common compression formats.

TABLE 1: Feasibility of file decompression for common compression formats.

Format	Algorithm	File extension	Stream possible?
UNIX compress	LZW	.Z	Yes
gzip	Deflate (LZ77+Huffman)	.gz or .tgz	Yes
zip	Deflate	.zip	Yes
7zip	LZMA	.7z	Yes
rar	LZSS	.rar	Yes
bzip2	BWT	.bz2	Yes (in blocks)
lha	LZ78+Huffman	.lha or .lzh	Yes
self-extraction	format-dependent	.exe	may be feasible

A file can be compressed more than once, i.e. recursively, and a compressed archive may contain multiple compressed files. It is difficult for on-the-fly decompressing to handle the recursive compression, because parsing the decompressed content continuously is needed to check if another compressed file exists. When the archive contains multiple compressed files with recursive compression, the communications between several decompressing and parsing processes are complicated. By contrast, the storage-based system can simply solve this problem by recursive decompression using the external program sequentially.

The stream-based system scans individual buffers where segments of file content are processed, but virus patterns may be across the segment boundaries. There are two solutions to this problem. The system can keep the scanning state of the virus scanner, i.e. which signature has its head matched the tail of the last segment and the matching position. Another solution uses a mechanism called cushioned scanning [15]. A cushioned scan extends the buffer with sufficiently large data from the tail of the previous scan buffer on the head side. That is, data in the cushion buffer is scanned twice. The size of a cushion buffer should not be shorter than the longest pattern in the virus database.

4. System Architecture

4.1 System Overview

Fig. 1 presents an overview of the stream-based system. The thin lines represent the direction of the user requests, while the bold lines represent the direction of mail transmission. The dispatcher intercepts the user requests and redirects them to the corresponding protocol handler. The protocol handlers include the SMTP and POP3 handlers. The directions of mail transmission in SMTP and POP3 are different. The attachments in a mail are encoded with MIME encoding, so a MIME parser is required. The MIME parser, decompression engine and virus scanner are all streamed-based. They can process the mail segment by segment. If no virus is found, the original data is forwarded as usual; otherwise, the proxy breaks the connection immediately and sends a notification to the user.

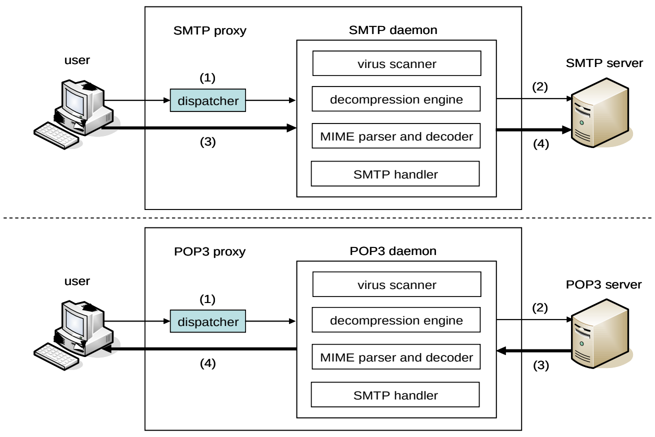

Figure 1. An overview of the stream-based proxy for the SMTP and POP3 traffic.

4.2 Processing workflow

The workflow of processing a mail is the same in SMTP and POP3. A typical mail is composed of the mail header, the mail body and the attachments (optional). Each component is processed sequentially. The mail body and the attachments are often MIME encoded. MIME encoded content includes a pair of MIME header and MIME body. The MIME body is encoded with an encoding method defined in RFC 2045 [15]. Common encoding methods are UUE, BASE64, quoted-printable, etc. The MIME header contains the information of the MIME body, such as the encoding method, the data type and the file names of the attachments. The following describe the processing of each component.

Processing the mail header

The SMTP protocol handler communicates with both the requester and the destination to initiate the transmission. When the mail transmission starts, the proxy reads a block of data from the socket buffer, and stores it in a raw buffer. The mail header is the first part of a mail. The header is read from the raw buffer to be checked if this mail is MIME encoded. If it is, the MIME parser is ready for parsing the MIME header and the MIME body.

Processing the mail body

A body parser can be added to check the body if it is a spam, or if it contains malicious links or JAVA/VB scripts. A spam mail is blocked, and the malicious scripts are removed. Since this work focuses on virus scanning, the mail body is simply forwarded to the destination. The body parser is not implemented in this work.

Processing the mail attachments

Attachments are mostly encoded and may be compressed. The MIME parser obtains the file name from the MIME header. According to the file name, the proxy processes the attachments in three rules. (1) The non-malicious files, like those with the extension "*.txt", can be ignored because they could not have viruses. (2) The files need to be scanned for viruses are those with the extensions such as "*.exe" and "*.doc". (3) If the extension suggests a compressed file, the file is decompressed first and the decompressed file is treated according to the three rules recursively.

5 System Implementation

5.1 Processing flow in the implementation

This system runs on a PC with Linux kernel version 2.6.10. It is implemented in Perl because of its outstanding string processing ability and various program libraries in Perl modules. Fig. 2 presents the processing flow in the implementation. The text in bold is the components in the system. The names in the parentheses are the existing open-source packages used in these components, all running within a single process in the user space. The arrows represent the relationship between these components. For example, the virus scanner interface calls *scanbuf* in the ClamAV shared library. Except that *Zlib* and ClamAV are shared libraries written in C, the other components are implemented in Perl.

When the kernel receives the packets, *netfilter* redirects the package with destination port 25 (for SMTP) or port 110 (for POP3) to the port the proxy listens to. The proxy accepts the connection and identifies a socket handler. After the SMTP handler communicates with the socket handler from the SMTP sender, it connects to the SMTP target to get another socket handler. With both the source and target socket handlers, a mail processor is created. The mail processor is written as a module created as an object at run time.

The mail processor handles the entire mail, including parsing MIME, reading the buffer from source socket, scanning the buffer and writing the buffer to target socket. The MIME parser in the mail processor does not use any existing open-source codes, but we refer to an open-source package Anomy, a stream-based MIME parser, to write our own. Because every connection creates a mail processor object, the mail processor becomes the main source in the memory consumption due to a large number of connections. The mail processor is independent of any protocol. To monitor the POP3 service, the POP3 handler instead of the SMTP handler is used.

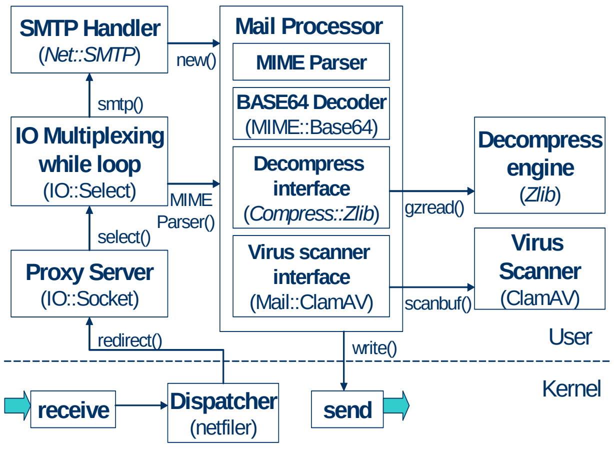

Figure 2. Processing flow in the implementation.

The text in the italic type means the codes of that package is modified for our purposes, including *Net::SMTP*, *Compress::Zlib* and *Zlib*. Because of I/O multiplexing, *Net::SMTP* is modified to process line by line whenever a socket is selected. *Compress::Zlib* is a Perl module and an interface to call the *Zlib* shared library in Perl. The original *Zlib* fails if it reads the end of data stream that is not equal to the end of file. The limitation is removed to support partial decompression. Other packages without modification can be upgraded to a newer version if the names and arguments of the function used in package remain their original definitions.

The system supports the file compressed by *Zlib* at first in our implementation. *Gzopen* and *gzread* are functions in the *Zlib* shared library. Because *Zlib* is designed to decompress an entire file, it opens a file handler by *gzopen* function before any decompressing by *gzread*. The system treats the handler *handler_out* as the file opened by *Zlib*, and inputs the decoded data into the handler *handler_in*, which connects with the handler *handler_out* by the inter-process communication mechanism called *Pipe*. The handler *handler_in* needs to be set as non-blocking I/O, thus making possible inputting the decoded data to the handler *handler_in* and reading decompressed data from the handler *handler_out* in turn. The combination of *handler_in*, *Pipe* and *handler_out* can be seen as a queue free for reading and writing at any time. This mechanism is applicable to any compression library originally designed to handle an entire file.

5.2 Single process concurrency

The proxy is implemented as a single process and use *select* to achieve concurrency because both multi-processing and multi-threading are inefficient and can consume huge memory. Because only one process handles all clients in turn, the state of every client is kept. Every time when I/O multiplexing selects a client to handle, the system calls the corresponding function according to the *state* of clients. Fig. 3 shows the set of states of a client during mail processing. Except that the SMTP and “quit or next” states are related to the SMTP protocol, other states are kinds of MIME parsing states. ”Bypass”, “scan” and “decompress” handle the attachments.

To achieve short response time, the processing time in each state should be short. The SMTP protocol handler handles one-line protocol message at a time in the SMTP state. The system reads only 8 KB each time when handling the three types of attachment. AMaViS, however, receives all mails and stores them in the disk first, and then processes mails sequentially. If there is a large file in front of many small mails, small mails need to wait until the large one has been finished. The average processing latency in the storage-based proxy may be long because the large mail blocks the small mail. The stream-based proxy can have short latency and service the clients fairly.

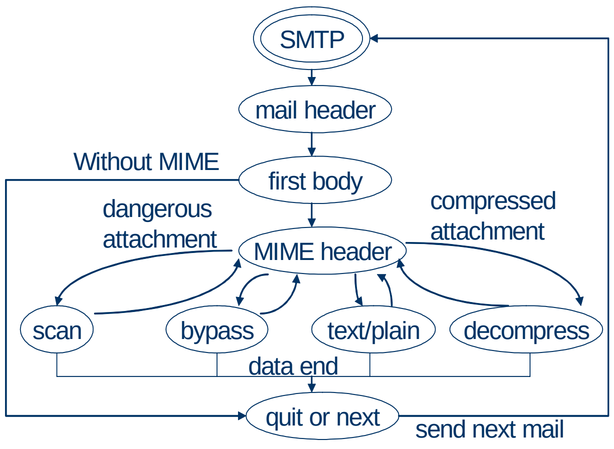

Figure 3. The states of mail processing.

6 Benchmarking

6.1 Test bed

We compare the stream-based mail proxy with a storage-based mail proxy, AMaViS. These two proxies are installed on a PC with 1GHz Pentium III CPU, 512MB SDRAM, 20GB hard disk and 100 Mbps Ethernet. The operation system is Linux with kernel version 2.6.10. Perl 5.8.5 runs both proxies since both are implemented in Perl. Both proxies use ClamAV version 0.83 as the virus scanning engine. Postfix serves as the MTA to work with AMaViS.

For fairness, AMaViS is configured in the following way. (1) The mail spam function is disabled since our stream-based proxy does not check the spam. (2) ClamAV runs in daemon mode that is faster than command line mode. (3) The cache mechanism is disabled in AMaViS. Two types of mail serve as the mail traffic in our benchmarking to represent different processing mechanism. The first is the mail with 1 MB executable attachment and will not to be scanned for viruses or decompressed. The proxy simply forwards this mail. The second is the mail attaching the compressed file from the above 1 MB file. The compression ratio is 37%. Both proxies have the same content to be scanned.

6.2 Performance and the impact of different mail content

The latency and throughput are measured in the external benchmarks. The latency is the time from the start of sending one mail to the end of receiving on target MTA. When the proxy is used, it holds the mail for a while. The latency with our proxy, AMaViS and no proxy environment are observed. Fig. 4 presents the improvement in latency. The latency is 102 ms without extra processing of the proxy. When the proxy function is involved, our proxy exhibits much shorter latency than AMaViS in all configurations. Our proxy takes 213 ms and 105 ms when forwarding a mail; AMaViS takes 1553 ms and 780 ms. Compared in virus scanning and decompressing, the latency of our proxy 518 ms and 527 ms, shorter than 1802 ms and 1267 of AMaViS.

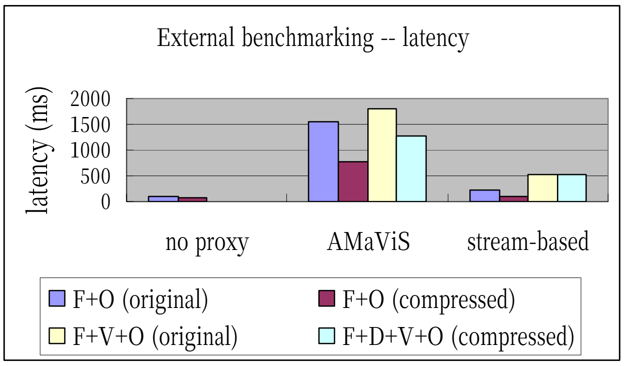

Figure 4. Latency of sending a mail.
(F: forwarding O: other mail processing V: virus scanning D: decompressing)

Throughput is defined as the total processed mail size divided by the elapsed time. A large number of identical mails are sent through the proxy and the total elapsed time is measured. The throughput of our proxy with simple forwarding is 65.2 Mbps, which is very close to the throughput of 69.93 Mbps without any proxy. AMaViS gets the throughput of 9.51 Mbps even when it disables both anti-virus and anti-spam functions. The storage-based architecture itself is a bottleneck.

Fig. 5 shows the throughput with virus scanning and decompression. With virus scanning but without decompression, our proxy has 21.79 Mbps. Dropping from 65.2 Mbps in simple virus scanning implies virus scanning is a bottleneck. AMaViS gets 6.9 Mbps with virus scanning, slightly dropped from 9.51 Mbps in simple forwarding. The notation "_E" denotes the effective throughput in scanning decompressed files. Because the file size is expanded after decompression, the effective throughput is higher than the throughput calculated from the original file size.

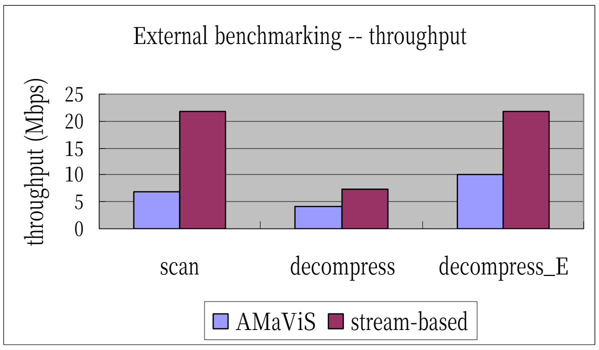

Figure 5. Throughput with virus scanning and decompression.

6.3 Buffer Requirement

We evaluate the buffer usage by monitoring the disk and memory usage of two proxies while there are a variable number of clients. Each client sends one mail attaching a 300KB file compressed from a 1 MB file. Fig. 6 presents the memory and storage usage of both proxies.

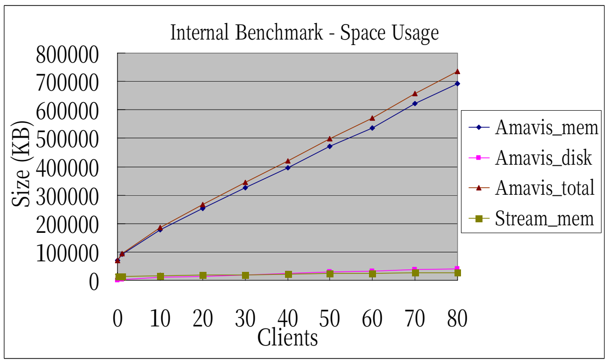

Figure 6. Space usage of memory and disk.

Amavis_mem and Amavis_disk denote the memory usage and disk usage, respectively. Because AMaViS need to cooperate with Postfix, both are counted. Amavis_total is the sum of Amavis_mem and

Amavis_disk. Stream_mem means the memory usage of our proxy. Since no temporary files are in the stream-based proxy, there is no disk usage. The storage-based proxy is shown to use much more space on both memory and disk than our stream-based proxy, which uses nearly constant memory space. The extra memory consumption in the stream-based proxy is the creation of the mail processor object, as discussed in Section 5.1.

In the internal bottleneck analysis, the execution time of each step in the mail processing is recorded. The steps include SMTP handling, MIME parsing, decompression, and virus scanning. Among them, virus scanning takes above 60% of the total execution time, while decompression takes only around 10%. This testing reveals that virus scanning is the bottleneck that should be accelerated in a better approach, such as hardware implementation or a better searching algorithm for virus signatures.

7 Conclusions and Future Work

This work designs and implements a stream-based mail proxy with interleaved decompression and virus scanning to prevent the storage of an entire mail. Without storing the entire mail, the file system access is eliminated and the buffer usage is saved. The external benchmarks prove the effectiveness of our stream-based proxy over the storage-based proxy, both in latency and throughput. In simple forwarding, the throughput of our proxy is 65.2 Mbps, while that of AMaViS is only 9.51 Mbps. In virus scanning, the throughput of our proxy is 21.79, while that of AMaViS is 6.9 Mbps. The disk space required is proportional to the number clients and the mail size in AMaViS. Our proxy shows nearly constant memory consumption compared with AMaViS. The internal benchmarking reveals virus scanning is the bottleneck.

This system is feasible at the embedded system environment without a hard disk and is more scalable than the traditional storage-based proxy. Anti-spam is another useful function in the mail proxy, and it can be added into a future version of the streamed-based proxy.

References

[1] Perl module -- Net::SMTP::Server, *http://search.cpan.org/~macgyver/SMTP-Server-1.1/Server.pm*.
[2] Clam AntiVirus system, *http://www.clamav.net*.
[3] Zlib, *http://www.gzip.org/zlib*.
[4] Perl module: Compress::Zlib, *http://search.cpan.org/~pmqs/Compress-Zlib-1.3.4/Zlib.pm*.
[5]AMaViS -- A Mail Virus Scanner, *http://www.amavis.org*.
[6] Trend Micro, *http://www.trendmicro.com*.
[7] Fortinet, *http://www.fortinet.com*.
[8] F-pod Antivirus, *http://www.f-prot.com*.
[9] CP Secure, *http://www.cpsecure.com*.
[10] The Anomy mail tools, *http://mailtool.anomy.net*.
[11] K. Wu, P.S. Yu and J.L. Wolf, Segment-based Proxy Caching of Multimedia Streams. *Proc. 10th Intl. World Wide Web*, Hong Kong, 2001, 36-44.
[12] C.H. Chi, J. Deng, Y.H. Lim, Compression Proxy Server: Design and Implementation. *Proc. 2nd USENIX Symp. Internet Technologies & Systems (USITS)*, Boulder, CO, 1999.
[13] Things you need to know before programming Perl ithreads, *http://www.perlmonks.org/?node=288022*.
[14] Y. Miretskiy, A. Das, C.P. Wright and E. Zadok, Avfs: An On-access Anti-Virus File System, *Proc. 13th USENIX Security Symposium*, San Diego, CA, 2004, 73-88.
[16] N. Freed and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, *RFC 2045*, 1996.