RESEARCH FEATURE

Designing an Integrated Architecture for Network Content Security Gateways

Ying-Dar Lin, Chih-Wei Jan, and Po-Ching Lin
National Chiao Tung University

Yuan-Cheng Lai
National Taiwan University of Science and Technology

Installing multiple network security products to detect and block viruses, spam, and other intrusions introduces substantial overhead in interprocess communications and kernel/user space interactions. Tightly integrating these functions in a single gateway can streamline the packet flows and dramatically increase Web and mail traffic throughput.

Most network content security systems that detect and block viruses, worms, Trojan horses, spam, and other intrusions are designed to operate independently or redirect traffic from cooperating devices. Increasing computer processing power has made it economically feasible to integrate multiple functions into a single gateway, but the industry remains divided over whether that is preferable to an all-in-one solution. Separate devices can be matched to specific functions and are more scalable and reliable; on the other hand, managing numerous heterogeneous systems can be costly in terms of both time and money.[1]

Researchers in this area have primarily focused on enhancing performance. For example, network security systems can be distributed for scalability—for example, by "slicing" traffic into manageable sizes.[2] In addition, packet content processing can be expedited. For example, various algorithms are designed to speed up packet classification,[3] while proposed signature-matching algorithms accelerate content inspection in both software and hardware.[4,5] Other work has examined architectural aspects such as software frameworks for more resilient firewall protection,[6] TCP splicing to hasten packet forwarding in an application proxy,[7] and multithreading capability.[8]

Little research, however, has examined integrated content security architectures.[9] Commercial offerings such as Astaro security gateways (www.astaro.com), Fortinet's FortiGate series (www.fortinet.com/products), and L7 Networks' firewalls (www.l7.com.tw/products.scan.eng.php) claim to integrate various functions such as intrusion detection and protection, spam blocking, and Web filtering. However, these products are black boxes with no way to study their inner workings.

To address this issue, we integrated five popular open source content security packages:

- Snort (www.snort.org) is a network intrusion detection and prevention system that captures packets on the network and analyzes their content for the signatures of various attacks and probes. The system can block as well as detect and log malicious traffic.
- ClamAV (www.clamav.net) is an antivirus toolkit that includes a virus scanner daemon and command, a library for developing virus scanners, and an up-to-date virus database.
- SpamAssassin (http://spamassassin.apache.org) uses various tests to identify spam signatures. Written in Perl, it works with numerous mail servers such as Sendmail and Postfix.
- amavisd (www.ijs.si/software/amavisd) is a program written in Perl that provides an interface between the mail transfer agent (MTA) and content checkers such as ClamAV and SpamAssassin.

Published by the IEEE Computer Society

- DansGuardian is a Web content filter that analyzes Web pages using phrase matching, URL filtering, and other techniques (http://dansguardian.org). It is mandated to work with a proxy such as Squid.

Using a set of external and internal benchmarks, we compared packet-flow performance during content inspection using a loosely integrated arrangement with a tightly integrated content security gateway.

Our research indicated that this new architecture substantially reduces overhead in interprocess communications and kernel/user space interactions. The study also revealed that the main bottlenecks of content processing are string matching in Web filtering and RAM disk access in mail processing.

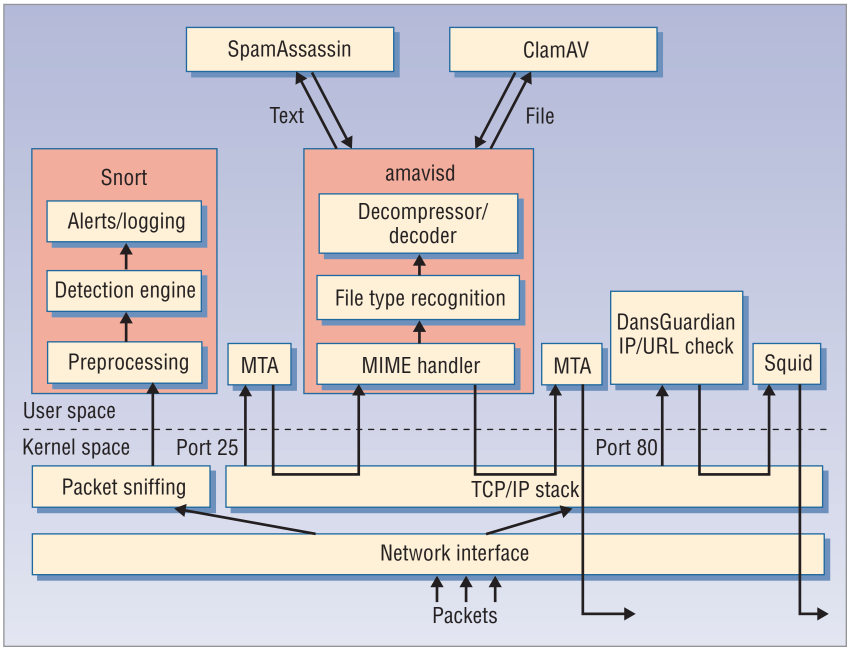

Figure 1. Components and packet flows of a loosely integrated content security architecture.

LOOSELY INTEGRATED ARCHITECTURE

Figure 1 shows the components and packet flows of a loosely integrated architecture, in which the five content security packages are simply installed. This approach introduces overhead, in the form of interprocess communications and duplicate kernel/user space interactions, which consumes system resources and reduces performance.

Components

Snort captures packets from the packet-filtering interface libpcap (http://sourceforge.net/projects/libpcap), executes a series of preprocessing steps such as packet reassembly and normalization, and sends these packets to the detection engine for signature matching. Snort generates alerts and can block and log any intrusion.

After receiving a message from the MTA, amavisd invokes the Multipurpose Internet Mail Extensions (MIME) handler to decode the mail and determine whether it contains any attached files. amavisd recognizes the file types and decompresses them if necessary. It then sends the decoded text to SpamAssassin and ClamAV. SpamAssassin analyzes the message for spam by matching it against a list of signatures, while ClamAV scans for viruses in the attached files. Based on the results reported back, amavisd decides whether to block the mail.

DansGuardian receives a request from the client and then matches the requested URL or IP address against its blacklist or whitelist. If the request is permitted, DansGuardian passes it to Squid; otherwise, it blocks the request. DansGuardian also analyzes the responses to decide whether to block the content according to its configuration.

Packet flows

When intrusion protection and Web-filtering functions are activated, HTTP traffic flow is as follows:

1. Snort captures packets from libpcap.
2. DansGuardian receives the request through the TCP/IP protocol stack at port 80.
3. DansGuardian checks whether the request is permitted.
4. If permitted, DansGuardian passes the request to Squid, which then connects to the Web server.

This process involves three user/kernel interactions (steps 1, 2, and 4) and one interprocess communication (step 4).

SMTP traffic flow is as follows:

1. Snort captures packets from libpcap.
2. The MTA receives the mail through the TCP/IP protocol stack at port 25.
3. The MTA forwards the mail to amavisd, which keeps the mail in the RAM disk.
4. amavisd calls SpamAssassin to check the mail.
5. amavisd sends the attached files to ClamAV for virus scanning.
6. amavisd forwards the mail back to the MTA.
7. The MTA relays the mail to the next MTA.

This process involves four user/kernel interactions (steps 1, 2, 3, and 6), two interprocess communications

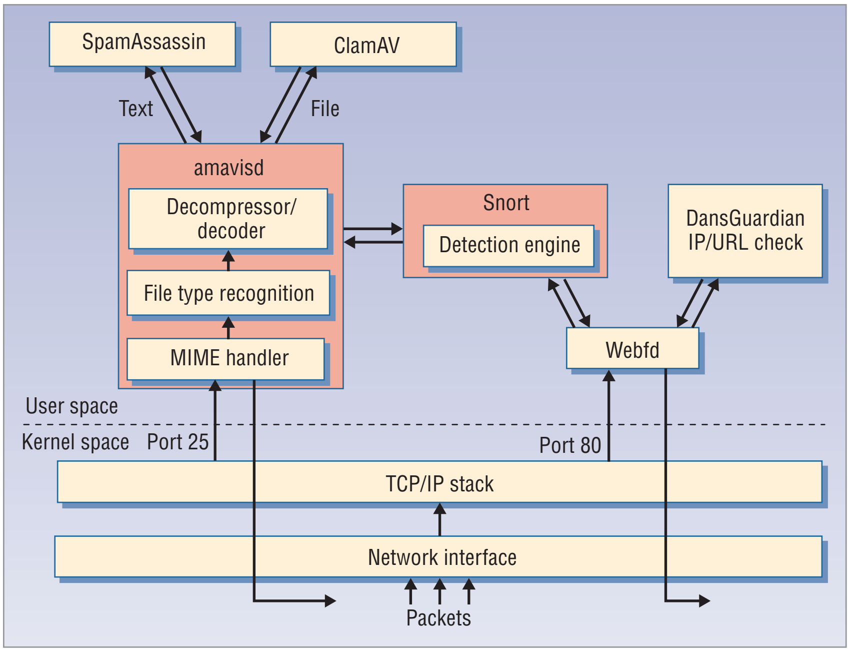

Figure 2. Tightly integrated content security architecture with new packet flows. The gateway uses a combination of multithreading and I/O multiplexing.

(steps 3 and 6), one process invocation (step 5), and one RAM disk access (step 3).

Overhead

The loosely integrated architecture incurs three types of overhead. First, interprocess communications between cooperating packages require additional packet copying between the kernel and user spaces. Second, amavisd and DansGuardian fork processes to serve multiple clients concurrently; context switching between these processes is expensive. Third, Snort captures packets from libpcap, but the other packages acquire them through the TCP/IP protocol stack—the system must copy packets from the kernel space to the user space and reassemble them individually in both packet flows.

TIGHTLY INTEGRATED ARCHITECTURE

To address these problems, we propose a more tightly integrated architecture, shown in Figure 2, that minimizes interprocess communications and eliminates duplicate kernel/user space interactions. To avoid process forking, the new gateway uses a combination of multithreading and I/O multiplexing. In addition, Snort functions as a shared library that derives content to be detected from the other packages.

Snort as shared library

To avoid redundant packet reassembly and packet copying, Snort source files are compiled with the option `-fPIC` for dynamic linking and made into a shared library with the command `ld-share`. An application program, say a proxy program, that needs intrusion detection and protection calls `fpInitDetectionEngine` to initialize the detection engine, and `CreateDefaultRule` to include default rules. The program fills the `Packet` structure with reassembled packet content from the TCP/IP protocol stack port 25 or from the raw socket and calls `fpEvalPacket` to analyze `Packet`. Snort returns detection results to the caller, which can block and log any intrusion.

The new architecture can be extended to detect and prevent nonapplication attacks by running a packet-capturing program that passes nonapplication packets, such as Internet Control Message Protocol packets or `SYN` (synchronize) segments, to the shared library. This program can refer to the related Snort rules, such as those for distributed denial of service, to do preliminary filtering to minimize packets passed to the shared library.

Stand-alone amavisd

To handle SMTP traffic, we modified amavisd to serve as a stand-alone mail proxy that can receive mail from port 25 (it is not intended to replace a typical mail server's full functionality). We also upgraded amavisd from a multiprocess proxy to a multithreaded one to be more scalable. We use multithreading instead of the `select` system call because the latter processes each incoming message sequentially, which degrades concurrency.

Two important modifications make amavisd a multithreaded stand-alone server without MTA support. First, we replaced the `Net::Server::PreForkSimple` module, which forks a new process to handle each incoming connection, with the `Net::Server::Thread` module, which implements a multithreaded server. Second, we changed the `mail_via_smtp_single` subroutine to relay checked mail directly to the next mail server rather than to the MTA.

ClamAV operates in the daemon mode to save the loading time of itself and its signatures during each invocation. amavisd communicates with the ClamAV daemon via the socket interface. SpamAssassin is still a PERL library residing in the memory. The amavisd interface also calls the Snort shared library to detect possible intrusions.

Webfd

To accelerate Web request processing time, we replaced DansGuardian with our own Web proxy, Webfd. Unlike DansGuardian, which forks processes to

serve new clients, Webfd uses the `select` system call to serve multiple clients concurrently.

DansGuardian's content-filtering component is compiled as a library that Webfd calls. The function `inBannedRegExpURL` checks whether the requested URL contains specified keywords in regular expressions, while `BannedURLList` and `inBannedSiteList` check whether the URL and site are permitted, respectively. Webfd calls these functions sequentially when processing the request. Webfd also calls the `checkme` function to determine whether the statistical score of the specified keywords exceeds the threshold and, if so, blocks the response.

Caching Web pages implies large disk storage. Many security devices are diskless, making caching infeasible. Our proposed architecture can fit in a diskless device, which is typical of a small office/home office gateway. Future work includes implementing caching capability in Webfd.

Our proposed architecture should be able to accommodate future revisions of the open source security packages.

Packet flows

HTTP traffic flow in the tightly integrated gateway is as follows:

1. Webfd receives the request at port 80 through the TCP/IP protocol stack.
2. Webfd checks whether the URL of the request is permitted.
3. Webfd calls the Snort library to check whether the request contains intrusion signatures.
4. Webfd makes a connection to the Web server if the request is permitted.

After integration, SMTP traffic flow is as follows:

1. amavisd receives the mail at port 25 through the TCP/IP protocol stack.
2. amavisd calls the Snort library to check if the SMTP message contains intrusion signatures.
3. amavisd calls SpamAssassin to check mail.
4. amavisd sends message to ClamAV to scan the attached files.
5. amavisd relays the mail to the next MTA.

Thus, the new architecture requires only one user/kernel space interaction (step 2) for both types of traffic and one interprocess communication (step 4) for SMTP traffic.

Integration cost

Our proposed architecture should be able to accommodate future revisions of the open source security packages, which are updated frequently. The architecture does not alter core functions such as `fpInitDetectionEngine` in Snort; they can therefore be extracted in the same way as in current versions as long as function names and parameters remain the same. The integration cost would only be high if the relevant functions have dramatic revisions, such as modified parameters or new APIs, which are not common in practice.

BENCHMARK RESULTS

To quantitatively assess the tightly integrated architecture, we compared its packet-flow performance during content inspection to that of the loosely integrated architecture using a set of external benchmarks. We also used several internal benchmarks to determine which security functions were most impacting processing time. We performed the tests on PCs with a 1-GHz Pentium III CPU, 256 Mbytes SDRAM, a 20-Gbyte hard disk, and the operating system NetBSD 1.6.1.

External benchmarks

For the HTTP benchmark, we used WebBench (www.veritest.com/benchmarks/webbench/default.asp) to measure the maximum requests per second and maximum throughput in megabits per second (Mbps). The setup included an Apache 1.3.12 Web server on one side of the gateway and eight PCs, each emulating 10 clients, running WebBench on the other side. The system determined whether requested Web pages were permitted by checking their text against a keyword database. The benchmark revealed differences in performance between the old and new architectures as well as the influence of URL blocking, keyword filtering, and intrusion detection functions.

For the SMTP benchmark, we modified Postal (www.coker.com.au/postal) to measure the maximum number of messages per second as well as maximum throughput in Mbps. We set up a mail server on one side of the gateway, while Postal acted as the mail client on the other side. We altered the original program, which only generates mail messages of random data, to support attaching files. The gateway determined whether each message, which contained one line of text and a 1-Mbyte file, was spam and whether the attached file contained a virus. The benchmark also revealed the influence of the antispam and antivirus functions.

HTTP test results. Web-filtering performance is proportional to response content size. A Web page is typically 4 to 8 Kbytes and rarely exceeds 40 Kbytes. Our test used both 5- and 40-Kbyte Web pages. Because Webfd does not yet have cache capability, we turned off Squid's caching function for fairness.

Figure 3a compares the maximum throughput of DansGuardian and Webfd, with and without Snort. Webfd with Snort improves performance by 83 percent for 40-Kbyte pages and more than doubles it for 5-Kbyte pages. The degradation by Snort in Webfd (0.48 Mbps) is smaller than that in DansGuardian (0.86 Mbps).

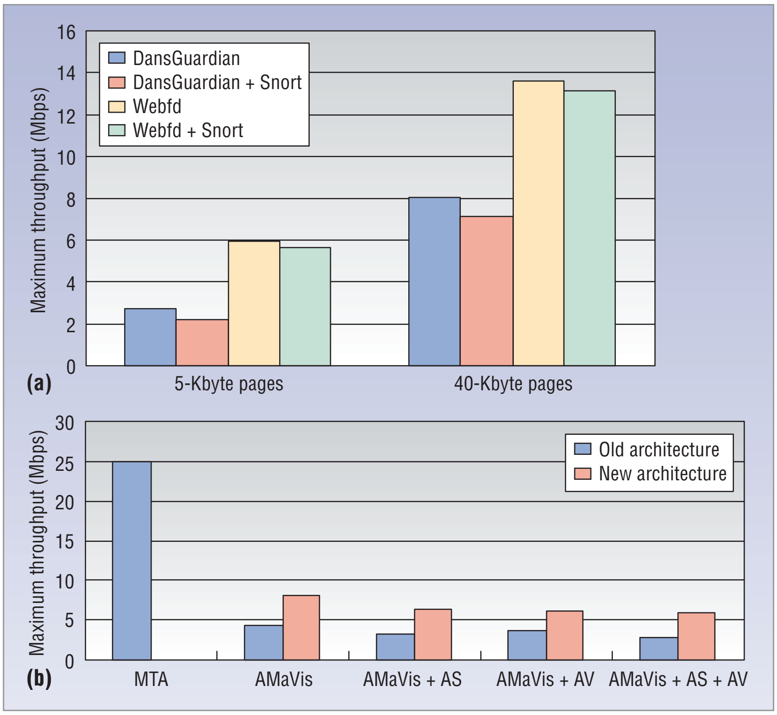

Figure 3. External benchmarks. Maximum throughput for (a) HTTP traffic with and without Snort and (b) SMTP traffic with and without antivirus (AV) and antispam (AS) functions enabled.

SMTP test results. Figure 3b compares the maximum throughput of the old and new architectures, with and without the antispam and antivirus functions enabled. When only the MTA relays the mail, maximum throughput achieves 25 Mbps. Passing mail through both the MTA and amavisd significantly degrades performance to 4.4 Mbps, even without enabling the functions. Because amavisd involves complex processing such as decoding and decompressing, its throughput is much lower than that of a pure MTA. Nevertheless, amavisd's maximum throughput doubles in every configuration of the new architecture.

Internal benchmarks

To assess the Web-filtering functions' relative impact on HTTP request and response processing, we measured the elapsed time of each code segment by inserting time stamps from the system call `gettimeofday` before and after the segment. The tool gprof records each function's elapsed time and number of calls during program execution. Chariot (www.netiq.com/products/chr/default.asp) generates the traffic, and fragroute (http://monkey.org/~dugsong/fragroute) breaks the packets into fragments to force Snort to reassemble them. For SMTP processing, we took time stamps from amavisd's built-in logging system.

HTTP test results. Figure 4a shows each function's HTTP request processing time percentage. `BannedRegExpURL` checks whether specified keywords in regular expressions are in the request, and `BannedURL` and `BannedSite` check whether the URL or Web site is in the banned list, respectively. Although `BannedRegExpURL`'s keyword database is much smaller than the databases for `BannedURL` and `BannedSite`, it consumes more than 60 percent of the processing time, compared to 24 percent for the other functions combined.

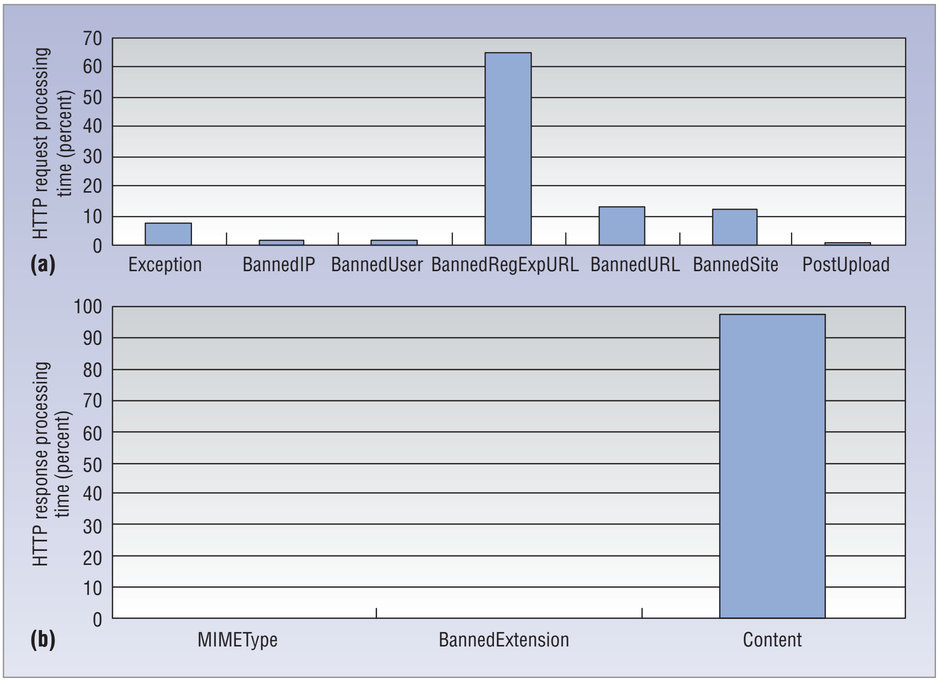

Figure 4. Processing time percentage of security functions in HTTP (a) request and (b) response processing. String matching is the primary bottleneck in Web filtering.

Checking specified keywords in the URL negligibly increased precision; we suggest turning off this function to accelerate request processing.

Figure 4b shows each function's HTTP response processing time percentage. `Content` checks whether the statistical score of specified keywords appearing in the response exceeds the threshold, and `MIME Type` and `BannedExtension` check MIME and file extensions, respectively. `Content` consumes more than 90 percent of the processing time, which is consistent with earlier findings in the literature on intrusion detection systems that string matching dominates the total processing for Web-intensive traffic.[10]

Figure 5 shows each Snort component's HTTP processing time percentage. String matching occupies the largest proportion and even exceeds 45 percent with 40- and 80-byte packets. Because the number of sessions is fixed, however, the relative proportions of session finding and rule operation decrease as packet length increases. TCP reassembly is smaller than anticipated and not a bottleneck in the detection phase.

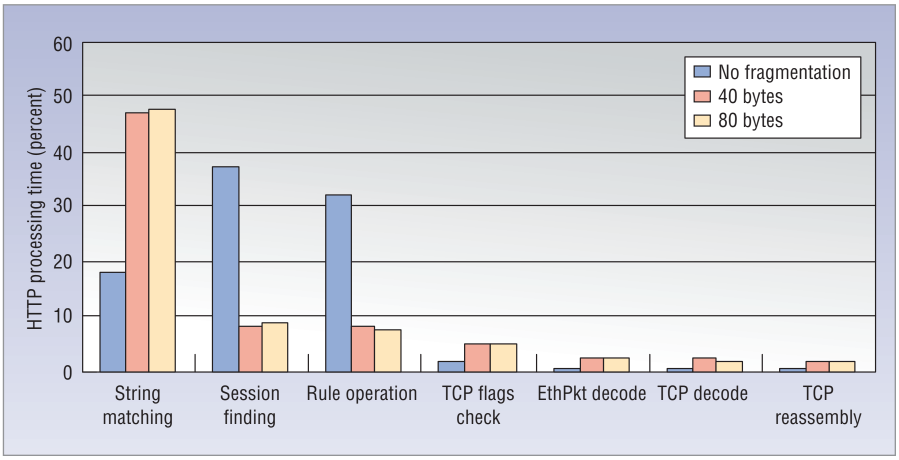

Figure 5. HTTP processing time percentage of Snort components. String matching occupies the largest proportion, while TCP reassembly is smaller than anticipated.

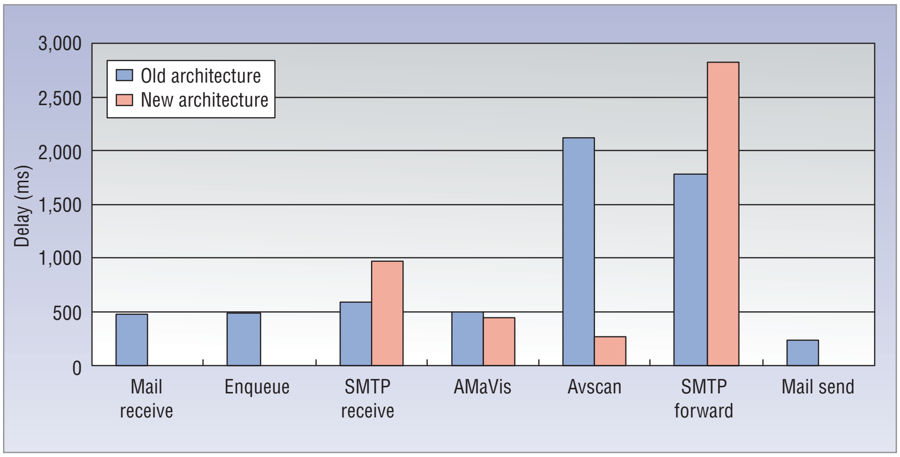

Figure 6. Latency of security functions in SMTP processing. Due to RAM disk accesses, SMTP receive and SMTP forward become bottlenecks in the new architecture.

SMTP test results. Figure 6 compares the old and new architectures in terms of the latency of each security function during mail-traffic processing. These functions include content decoding and file decompression (amavisd) and invocation of the external virus-scanning engine (Avscan).

The new architecture runs the scanning daemon in memory instead of invoking an external program, significantly reducing the time required to load signatures. Moreover, making amavisd a stand-alone proxy eliminates the need for mail-server processes including mail receive, `Enqueue`, and mail send. However, because amavisd must receive and send the mail itself, the SMTP receive and SMTP forward processing time increase by 21 and 62 percent, respectively. These two functions become the new bottlenecks.

Because the mail's content size is much larger than the request size, mail must be saved before further processing. The remaining processing, decoding, decompression, virus-scanning, and spam-checking operations all involve RAM disk accesses. SMTP processing throughput is thus much lower than in HTTP processing.

Tightly integrating open source content-security packages substantially reduces overhead between the software components and the kernel. Our tests indicate that modifying the Web filter to use I/O multiplexing improves throughput by 83 percent and that substituting multithreading for multiple processes in antivirus and antispam functions roughly doubles throughput.

In addition, we have determined that the main content-processing bottlenecks in a security gateway are string matching in Web filtering and RAM disk access in mail processing—in our tests, these occupied 48 and 62 percent of the processing time, respectively. These results prompt research on ways to further increase processing efficiency.

Our architecture can be extended to proxies of other traffic types. Some software components can also be improved to be fully functional. For example, in the future we plan on revising Webfd to support caching capability and upgrading amavisd to support on-the-fly streaming operation in memory. ■

Acknowledgments

This work was supported in part by the Taiwan National Science Council's Program of Excellence in Research, and in part by grants from Cisco and Intel.

References

1. K. Tolly and C. Bruno, "Measuring the Value of Integrated Perimeter Security," white paper, The Tolly Group, July 2004; www.astaro.com/firewall_network_security/why_integrated_security.
2. C. Kruegel et al., "Stateful Intrusion Detection for High-Speed Networks," *Proc. 2002 IEEE Symp. Security and Privacy,* IEEE CS Press, 2002, pp. 285-293.
3. P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network,* vol. 15, no. 2, 2001, pp. 24-32.
4. M. Norton, "Optimizing Pattern Matching for Intrusion Detection," white paper, Sourcefire Inc., 2004; http://docs.idsresearch.org/OptimizingPatternMatchingForIDS.pdf.
5. S. Dharmapurikar et al., "Deep Packet Inspection Using Parallel Bloom Filters," *IEEE Micro,* vol. 24, no. 1, 2004, pp. 52-61.
6. R. Knobbe, A. Purtell, and S. Schwab, "Advanced Security Proxies: An Architecture and Implementation for High-Performance Network Firewalls," *Proc. DARPA Information Survivability Conf. and Exposition,* vol. 1, IEEE CS Press, 2000, pp. 140-148.
7. O. Spatscheck et al., "Optimizing TCP Forwarder Performance," *IEEE/ACM Trans. Networking,* vol. 8, no. 2, 2000, pp. 146-157.
8. D.C. Schmidt, T. Harrison, and N. Pryce, "Thread-Specific Storage for C/C++: An Object Behavioral Pattern for Accessing Per-Thread State Efficiently," tech. report 97-34, Washington Univ., Sept. 1997; http://st-www.cs.uiuc.edu/users/hanmer/PLoP-97/Proceedings/schmidt.tss.pdf.
9. Y-D. Lin, H-Y. Wei, and S-T. Yu, "Building an Integrated Security Gateway: Mechanisms, Performance Evaluations, Implementations, and Research Issues," *IEEE Comm. Surveys and Tutorials,* vol. 4, no. 1, 2002; www.comsoc.org/livepubs/surveys/Public/2002/Dec/wei.html.
10. S. Antonatos et al., "Performance Analysis of Content Matching Intrusion Detection Systems," *Proc. Symp. Applications and the Internet*, IEEE CS Press, 2004; www.ics.forth.gr/carv/papers/2003.SAINT04.idsperf.pdf.

Ying-Dar Lin *is a professor in the Department of Computer Science at National Chiao Tung University, Hsinchu, Taiwan. His research interests include design, analysis, implementation, and benchmarking of network protocols and algorithms; wire-speed switching and routing; and embedded hardware-software codesign. Lin received a PhD in computer science from the University of California, Los Angeles. He is a member of the IEEE and the ACM. Contact him at ydlin@cis.nctu.edu.tw.*

Chih-Wei Jan *is an MS student in the Department of Computer Science at National Chiao Tung University. His research interests include network security, content networking, embedded system design, high-speed networking, and performance evaluation. Jan received a BS in computer science from National Chiao Tung University. Contact him at cwjan@cis.nctu.edu.tw.*

Po-Ching Lin *is a PhD candidate in the Department of Computer Science at National Chiao Tung University. His research interests include network security, string-matching algorithms, hardware-software codesign, content networking, and performance evaluation. Lin received an MS in computer science from National Chiao Tung University. Contact him at pclin@cis.nctu.edu.tw.*

Yuan-Cheng Lai *is an associate professor in the Department of Information Management at National Taiwan University of Science and Technology, Taipei, Taiwan. His research interests include high-speed networking, wireless network and network performance evaluation, Internet applications, and content networking. Lai received a PhD in computer science from National Chiao Tung University. Contact him at laiyc@cs.ntust.edu.tw.*