Global Information Assurance Certification Paper

Protecting Microsoft Internet Information Server using filtering and reverse proxy techniques

SANS Security Essentials
GSEC Practical Assignment

Version 1.3

Chris Roberts

Contents

Introduction **3**

Filtering **3**
Overview **3**
Parameters used in filtering **3**
Benefits **4**
Limitations **5**

Examples of Filtering-Based Solutions **5**
SecureIIS **5**
URLScan **6**
Snort **7**

Reverse Proxies **9**
Overview **9**
Benefits **9**
Limitations **10**

Examples of Reverse Proxy-Based Solutions **10**
Volera Excelerator **10**
Squid Web Proxy Cache **11**

Summary **12**

References **13**
Online Resources **13**
Software **14**

Introduction

Since the CodeRed and Nimda Internet worm epidemics last year, Microsoft Internet Information Server (IIS) has been under intense scrutiny, both from Micr osoft itself, and the IT industry. It has been estimated that these two worms could have caused up to $3.25 billion of damage [1]. With these kinds of figures, it is no surprise that Microsoft has launched their 'Trustworthy Computing" initiative [2], to try to improve the security record of their products. With Microsoft working to make IIS a more secure product, and the release of their HFNetChk tool (which can inform administrators of missing patches), and the IISLockDown tool (uses best practices to configure IIS to be secure) [3], IIS web server administrators may be starting to feel that they can relax a little. However, evidence from DShield.org would suggest that the number of network probes directed at web servers remains high. Last month, 39.38% of all incidents recorded by DShield.org had destination port 80 (HTTP) [4], so were almost certainly directed against web servers.

Regardless of how well-secured an IIS system can be, it is a difficult job to take into account future attacks and vulnerabilitie s. Following guidelines to secure IIS, such as Microsoft's IIS Baseline Security Checklist [5], or Ben White's guidelines [6], will help to reduce the risk of vulnerabilities being successfully exploited, but cannot guarantee that the system will not be compromised by an future exploit. As an addition al measure, filtering or a reverse proxy can help the web administrator reduce this threat, since these techniques can be used to prevent non - legitimate requests ever reaching the webserver.

The aim of this pap er is to introduce the concepts of filtering and reverse proxies, and outline some of the products which can be used to accomplish this. I shall look at each of these techniques in turn, and products which can make use of them, to protect IIS webservers.

Filtering

Overview

Filtering is the use of a network device or software, to intercept incoming requests to the webserver, and pass them through filters, to decide whether the request is legitimate or not. This technique is very similar to intrusion de tection systems, or protocol -aware firewalls, where lists of rules are used to filter out suspect traffic.

Parameters used in filtering

Typically, filtering might use the following parameters, to decide whether traffic is legitimate:

- Well-formed HTTP request
  Whether or not the request conforms to the RFC 2616 for HTTP protocol traffic [7]. This would include standards such as permitted requests, header lengths, permitted characters, authentication process , etc.

- Well-formed TCP/IP packet
  Filters can detect malicious badly formatted packets, such as the use of fragmentation attacks (e.g. WinNuke), bad length packets, incorrect TCP flag combinations, incorrect TCP handshaking (SYN, SYN/ACK, ACK) and other breaches of RFC 0791 (IP) and RFC 0793 (TCP) [8].

- Length of URL requested
  This will depend largely on the type of site, as dynamic or membership -based sites may require longer URL's. However, the web administrator may well be able to decide on the maximum length of a legitimate URL. Anything longer than this could be a potential buffer overflow.

- Length of parameters in request
  Similar to above. Any overly long parameters passed could be attempts to exploit a buffer overflow in the web server process (e.g. active server page, CGI script) which is responsible for processing the data.

- Frequency of requests
  This filter uses statistics to detect suspicious activity. If a particular IP address sends many requests for the same URL, or where it makes many requests in a short space of time, this could be seen as an attack on the webserver. This particularly applies to denial of service attacks, and automated scans by Internet worms.

- Pattern matching known attack patterns
  This is the most obvious type of traffic to filter. Where attacks by Internet worms, or malicious exploits have been analysed, and the signature is known, then blocking them will reduce load on the webserver, and mitigate the risk of the IIS system not having been correctly updated. This approach is also very effective where no security updates exist for the vulnerability, since it relies only on the signature of a possible attack being known. Organisations such as Incidents.org [9] *and SecurityFocus' ARIS Threat Management System* [10] *monitor distributed data from intrusion detection systems, and act as early warning systems for new types of web attacks, such as Internet worms.*

Benefits

The major benefit of using filtering is that it prevents attacks reaching the IIS server, whether it is vulnerable or not. Whilst this cannot replace host -based system hardening, it can mitigate the risk where an IIS server has been incorrectly configured. This is a valuable change of emphasis, where an organisation has staff skilled in network -based security, but has less expertise in the area of IIS, and Windows. In this kind of scenario, there is also the added benefit that the filtering system need not run a Windows -based OS at all, and so would not be vulnerable to attacks on the Windows OS itself.

Another major advantage of using filtering is its ability to pre-empt new exploits and worms, based on the familiar patterns of existing attacks. In particular, buffer overflows can all but be eliminated, if filters have sufficiently aggressive limits on URL and parameter length.

Logging data collected during the course of filtering a webserver can also be very valuable. Logs provide patterns in themselves, such as repeated attack attempts from the same host, and new attack signatures obtained from the limit-based filters discussed above.

Limitations

Filtering is only effective where rules can be clearly defined, which identify suspicious or malicious traffic. The main drawback of filtering is that it is not heuristic – it cannot usually adapt filters to detect attack patterns, they must be pre-defined. Using statistics, to look for suspiciously high numbers of requests or repeated requests, from a single host, does give a basic level of prediction, but in practice, few actual implementations of filtering use this heavily. Where IP-spoofing is used, collecting statistics becomes an even more difficult task.

A drawback of using pre-defined filters is that this approach can lead to false-positives, which leads to legitimate requests for content from the webserver being blocked. The filters must be continually fine-tuned, according to the type of content on the webserver, and the number and rate of requests expected. This requires that the administrator of the filtering system knows the websites they are protecting well – so they can include sensible parameters for the filters.

Where a webserver is under heavy load, filtering may introduce latency in the rate at which requests can be processed, since the filtering adds another layer at which the request needs to be processed. The filtering operation needs to perform network layer pattern-matching, content reconstruction, and content-based pattern-matching before the request can be deemed legitimate and passed on to the IIS webserver.

Finally, if a webserver uses SSL to encrypt traffic between the browser, and the IIS webserver, then a disadvantage of this can be that filtering systems which are not installed on the IIS webserver host will not be able to decrypt the HTTP traffic to filter the content.

Examples of Filtering-Based Solutions

Having broadly introduced and discussed the concept of filtering, and how implementations might work, I'll now move on to highlight some specific implementations of filtering techniques, and the features they have.

SecureIIS

SecureIIS is a host-based application firewall product produced by eEye Digital Security [11]. It works as an application, installed on the IIS server itself, and uses many of the filtering parameters described earlier, to decide whether requests made to the IIS server are valid or not. SecureIIS mainly uses attack pattern matching, based on request content, and length limits on various request parameters, to detect suspicious traffic, and prevent buffer overflows.

A screenshot of the configuration interface may be seen below :

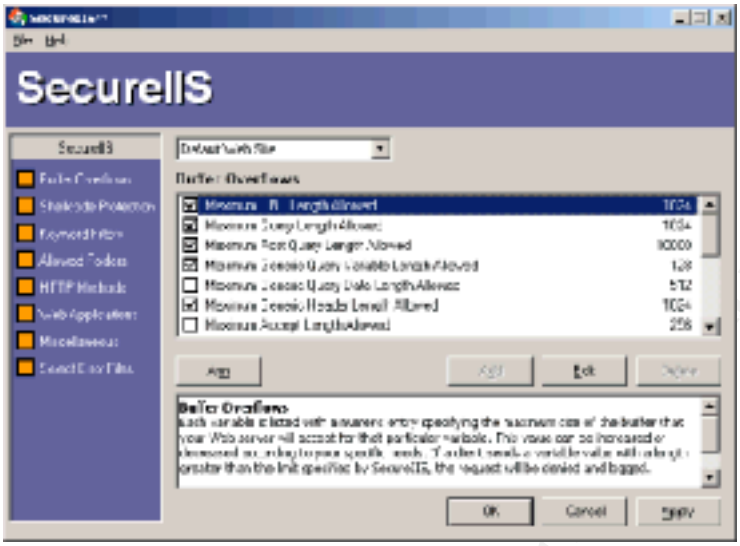

Fig1: SecureIIS configuration screenshot (from the eEye - SecureIIS product information website)

Once SecureIIS detects a suspicious request, it replies to it, on behalf of the IIS webserver, by returning an error webpage, which informs the potential attacker that their request has been denied.

SecureIIS can also receive updates from eEye, so that future attack signatures can be incorporated into the pattern matching engine. There is also provision for adding custom filters, which will pattern match on the request content, to give the product extra flexibility.

As SecureIIS is installed on the same host as the IIS webserver, and integrates into IIS, it can also protect the webserver against SSL-encrypted traffic, something which network-based filtering solutions would be unable to do.

The strength and weakness of this product lies in the fact that it is host-based. Its tight integration with IIS, and its ability to protect against SSL-encrypted HTTP requests are a benefit of this. However, the load incurred by processing and filtering each request is handled on the webserver itself, which is a disadvantage for a busy webserver. In addition, the webserver remains vulnerable to OS-based attacks, which are not protected by SecureIIS.

This product seems to be a very good solution for a web administrator who is relatively new to IIS. The buffer overflow parameters are preset, and the system will automatically be protected against known attack signatures. eEye have done some excellent research into the CodeRed worm [12], and the inclusion of an auto-update feature could be very valuable. I could see problems however, if the site received lots of requests which exceed the default maximum length parameters. SecureIIS could require a considerable amount of tuning to be effective at preventing buffer overflows, without blocking legitimate requests.

URLScan

As part of Microsoft's security initiative, they have released their own IIS request filtering tool, URLScan [13]. Again, the software is host-based; being tightly integrated with IIS, and similarly it intercepts HTTP requests, filtering them for known attack patterns.

Unlike SecureIIS, the URLScan product is not feature-rich, but this is unsurprising, since unlike SecureIIS, it is free. URLScan is simply a filtering mechanism for pattern matching, and blocking certain types of requests. It breaks down the filtering operation into four levels:

- Verbs
  Types of requests such as PUT, GET, POST, etc.

- Extensions
  File types such as .asp, .exe, .bat, .shtm, etc.

- Headers
  Certain types of HTTP header values are not permitted, e.g. Translate, Lock-token

- URL Sequences
  Any request pattern defined by the administrator, e.g. cmd.exe

In addition, URLScan will normalise the URL before performing pattern matching. This ensures that any encoding of the URL, to attempt to disguise an attack, will fail.

The URLScan tool is only useful where the web administrator has a lot of knowledge about the site, and can accurately predict what kinds of requests would be valid, and so deny all others. The IISLockDown tool can be used to install URLScan with a sample template, tailored for the webserver's primary function, e.g. Exchange 5.5 Outlook Web Access Server, or SharePoint Portal Server.

This tool is a valuable addition to a web administrator's toolkit, but the only documentation available is a basic Readme-style document included in the distribution. This lack of documentation limits its use to experienced IIS web administrators. The templates provide the best tutorial on configuring URLScan. If Microsoft continues to develop this tool, and add more features, it could evolve to become an invaluable tool.

Snort

Snort is a free, open-source intrusion detection system (IDS) [14]. It uses pre-defined rules to filter network traffic, and issue alerts when suspicious traffic is encountered. This product is not targeted specifically at protecting web servers, nor does it block suspicious network traffic by default, but there are features in the software which easily support its use in this way. It can be run on either a Unix-based, or a Windows system.

Snort works by capturing incoming and outgoing network traffic, and matching these IP packets against signatures defined in its rules files. To capture traffic, it uses a network interface in promiscuous mode. In addition, the network device to which it is attached must be configured to duplicate network traffic for it. Possible solutions for duplicating the traffic might be the use of a router with port-mirroring, or a vampire tap. The rules can match on many parameters of the IP packet, including:

- Source or destination IP address
- Source or destination port

- IP protocol
- TCP flags set (where protocol is TCP)
- Packet length
- Packet fragmentation
- Packet contents

This would allow the definition of almost any malicious packet. The default rules included with the source distribution of Snort have a rules file dedicated to IIS exploits (web-iis.rules). An example rule, which would match CodeRed worm attacks, is shown below:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB -IIS ISAPI .ida access"; uricontent:".ida"; nocase; flags:A+; reference:arachnids,552; classtype:web -application -activity; reference:cve,CAN -2000 -0071; sid:1242; rev:2;)

In addition, Snort has a number of pre -processors, one of which can be used to decode HTTP traffic in the same way as URLScan, so that encoded requests are normalised before being compared to rules. Ot her pre-processors can reassemble fragmented IP packets, reassemble TCP traffic streams, detect port scans, and even detect statistical anomalies in traffic. The last two require configuring with parameters appropriate to the web administrator's environme nt. This might be difficult for web administrators who are not very familiar with IP networking concepts, but if correctly set up, provides an excellent early warning system in addition to the signature -based filtering.

Although the default Snort rules a re configured only to issue warnings when suspicious traffic is detected, it is possible to configure Snort to use the FlexResp component, to reject suspicious traffic. If FlexResp is used, the Snort system can generate responses to suspicious packets, including sending TCP RST packets to both the source and destination IP addresses. The possible responses are shown below (taken from the Snort Users Manual [15]):

- rst_snd - send TCP-RST packets to the sending socket
- rst_rcv - send TCP-RST packets to the re ceiving socket
- rst_all - send TCP_RST packets in both directions
- icmp_net - send a ICMP_NET_UNREACH to the sender
- icmp_host - send a ICMP_HOST_UNREACH to the sender
- icmp_port - send a ICMP_PORT_UNREACH to the sender
- icmp_all - send all above ICMP pack ets to the sender

Using FlexResp, Snort is able to effectively block potential attacks, by resetting the attacker's network connection to the IIS server.

Snort is a powerful IDS, with many features which would be useful for protecting both IIS itself, and the system from OS -based attacks. The pre -processors provide a level of protection in addition to filtering, using basic statistical analysis techniques. Snort can be run on a Unix or Windows system, so the platform can be chosen according to the reso urces, and skills available.

The drawback, from an IIS web administrator's perspective, is that the administrator of the system would need to have experience with IP networking concepts, to properly tune the Snort system to the network environment. Snort is also prone to false -positives, which can be a real problem if FlexResp is used to reset connections, where the traffic seems suspicious. The default

rules need to be modified to minimize these false-positives, since legitimate traffic may be potentially blocked by FlexResp. The safest solution is to run Snort over a period of weeks, and analyze the alerts produced, to best assess which rule entries produce false-positives, and how introducing FlexResp might affect the network. It would also be possible to customize the rules to only apply FlexResp responses to suspicious traffic bound for the IIS system(s) the web administrator wished to protect. The default rules allow the definition of a $HTTP_SERVERS variable, which defines the IP addresses which will be monitored for incoming web-based attacks.

Reverse Proxies

Overview

Reverse proxies are used to provide an alternative hosting platform for a website, where it is undesirable for the actual web server providing the content, to host the site itself. A reverse proxy caches the entire contents of the website, and clients connect to the reverse proxy to retrieve content. A reverse proxy may be used for a number of reasons, including for performance reasons, or for security reasons.

Benefits

The most obvious benefit of using a reverse proxy is that the original IIS web server containing the content is never accessed by clients, or exposed directly to the Internet. Whilst this does not mean that the IIS web server should not be well-secured, it does mean that it is only exposed to internal threats, since the reverse proxy provides the publicly accessible interface for the website.

There can also be performance benefits from using a reverse proxy. If a reverse proxy device is used to cache a slow web server, it may be faster at serving content, than the web server would be, if it received requests directly. Reverse proxy devices can also be used in load-balancing configurations, to further improve the speed of the website. Where an organisation has limited funds, they can use a high performance reverse proxy configuration to host all their publicly accessible web sites. This reduces the performance and redundancy requirements for the web servers themselves.

Using a reverse proxy configuration in this way also means that it is easier to protect the web sites, since the perimeter firewall needs only to permit HTTP connections to the reverse proxy system, and not the individual web servers directly. If filtering is used, as discussed earlier in this paper, then it can be easier to monitor the reverse proxy, to detect malicious traffic, rather than multiple web servers, on different points of the network.

Finally, the reverse proxy need not be based upon the same platform as the web server, and so can be chosen so that OS-based exploits could not compromise both the reverse proxy system(s) and the web servers themselves. Again, this may be useful in an environment where experience with administering IIS web servers is rather limited.

Limitations

Reverse proxying is most effective where the websites being cached have a relatively static content. Where the website is highly dynamic (e.g. generated on-the-fly from a database) it can be difficult to maintain a current copy of the site in the cache. In some cases, where content is highly time-dependant (e.g. share prices), it may even be impossible. This factor must be taken into account when the web administrator considers whether the site is suitable for reverse proxying.

As discussed, a reverse proxy configuration can be used to economically cache and host all the publicly accessible websites published by the organisation. Although this provides a single point for filtering, and allowing access through perimeter defences, it also provides a single point of failure – or attack through hacking and denial-of-service attacks. If the reverse proxy can be compromised, it may provide the intruder with further access to the source IIS webservers, which must be accessible to the reverse proxy since it caches their content. This risk can be mitigated, however, with the use of purpose-built reverse proxy devices, which often run cut-down versions of commercial operating systems, and so provide more limited scope for compromise.

Examples of Reverse Proxy-Based Solutions

There are large numbers of commercial reverse proxying solutions available produced by respected vendors, including Cisco, Microsoft, and IBM. It is beyond the scope of this paper to review them all. Instead I will consider two very different solutions, to give some idea of the options available.

Volera Excelerator

The Volera Excelerator product line [16] is a typical example of a purpose-built device, which can perform forward or reverse proxying in a load-balanced configuration. The device is installed in the network in the following way:

Fig 2: Volera Excelerator configuration (from the Volera Excelerator product information website)

The Excelerator device is based upon the Novell Netware operating system, and is configurable via web, telnet and FTP interfaces. It is capable of proxying FTP in addition to HTTP content, and Volera claim it is capable of responding to 2400 requests per second, at Gigabit line speeds.

The major feature of this product line is that it is also capable of reverse proxy caching streaming content, and HTTPS-secured sites, with the Media Excelerator and Secure Excelerator products.

The Secure Excelerator product is a particularly interesting concept – it can be used to transparently provide HTTPS security for a website, where the webserver itself does not handle encryption. Clients browse to the Secure Excelerator device, using HTTPS, which then decrypts the content, and forwards the request to the webserver it is reverse proxying, using HTTP. The response from the webserver is sent to the Excelerator device, which recodes all the links to HTTPS format, and then passes the resulting page data back to the client using HTTPS. This has two benefits – firstly, the overhead of encrypting and decrypting the data is handled by the Secure Excelerator device, instead of the webserver. This could be of real benefit to an organisation making heavy use of e-commerce, which requires secure transactions. Also, the incoming requests are decrypted by the Secure Excelerator, and forwarded on to the webserver using HTTP, which is not encrypted, and so requests could be filtered using any of the products described earlier.

URL filtering and proxy bypass features ensure that the reverse proxy can be disabled for certain websites, or areas of sites, which perhaps have frequently-changing dynamic content.

The Volera Excelerator products are easily configurable, and relatively simple to deploy, with no knowledge of the underlying system being required. The only criticism would be that purpose-built solutions do not always provide the most flexible solution, and have limited scope for dealing with unusual scenarios.

Squid Web Proxy Cache

At the other end of the market, the Squid Web Proxy Cache [17] is a free, open-source, Unix-based software web proxy. Part of the project is a series of extensions to enhance Squid's feature-set as a reverse proxy [18].

Currently the project is in a beta testing stage, but already the product can support the following: URL rewriting, load balancing, and the definition of source webservers.

The Squid software is highly configurable, with almost all aspects of the cache's performance being configurable by the administrator. The cache size, object timeout algorithms, and complex access control lists can all be defined.

In addition, Squid supports a number of highly useful reverse proxying features. It is able to communicate with other Squid caches, to work in a load-balancing configuration. Squid also supports 'delay pools' which can be used to define the maximum request-rate, or number of connections that are permitted at any time. This will prevent the webserver providing content from being swamped with requests, and might be useful in preventing a denial-of-service attack.

The base distribution of Squid supports a basic version of reverse proxying, which allows incoming HTTP requests to be rewritten on-the-fly, before being forwarded on to the web server providing content. This would mean that Squid could perform basic load-balancing for webservers, and also cache content returned by the webservers. This type of reverse proxying might well be adequate for webservers providing largely static content.

Squid uses complex access control lists (ACL's) to control who can access the cache, and what they can access. This is very useful for performing basic filtering, using the following:

- src - client IP address.
- srcdomain - client domain-name
- dstdomain - destination (webserver) domain
- time - time of day, and day of week
- url_regex - any matching pattern in the URL being requested
- port - destination (webserver) port address
- proto - transfer protocol (Squid can proxy HTTP and FTP)
- browser - any pattern matching the request's user-agent header
- req_mime_type - any pattern matching on the request content-type header
- arp - the client's Ethernet (MAC) address matching

Using these parameters, almost any combination of access requirements could be defined.

The Squid web proxy cache is a powerful piece of software, with a huge range of features and options. As a reverse proxy, it is a very immature product, but I expect it will improve quickly, and it will be a significant offering in the future. At the current time, it would be a considerable undertaking for an IIS web administrator to correctly configure a Squid reverse proxy cache, although it would be useful for an organisation with limited funds, or extensive experience with Unix-based platforms. At the current time it would only be suitable for reverse proxying websites with static content, using the 'acceleration' features.

Summary

The concepts outlined in this paper would be useful for protecting any web server, regardless of platform, or the software used to host the site. The examples focussed on Microsoft IIS, since recently a number of filtering solutions have become available for this software, and I wished to explore these products further.

The examples solutions show that there is no ideal product for every scenario. The resources and skills available in the organisation, as well the funds available will all be contributing factors. In general, the free, open source solutions considered seemed to be more difficult to set up and configure initially, if they are to be effective. However, once running, projects based on the Snort or Squid solutions would have more flexibility, and scope for future development.

When considering using filtering and reverse proxying together to protect an IIS webserver, it is important to consider whether the site will use HTTPS, and to choose a reverse proxy capable of handling HTTPS requests, so that the requests are sent unencrypted to the IIS webserver providing the content. Filtering products cannot monitor HTTPS sites – since the data contents are encrypted between the client and webserver, so the requests cannot be viewed by the filtering software.

Using filtering or reverse proxy techniques to protect an IIS webserver provides a proactive line of defence in addition to host-based hardening. These techniques can help to protect the webserver against more generic security threats and typical IIS exploits for undiscovered vulnerabilities.

References

Online Resources

1. Computer Economics, " 2001 Economic Impact of Malicious Code Attacks", 2002.
   URL: http://www.computereconomics.com/cei/press/pr92101.html

2. CNET Networks, Inc. , "Gates memo: "We can and must do better" ", January 17th 2002.
   URL: http://news.com.com/2009-1001-817210.html?legacy=cnet

4. DShield.org, "Top 10 Target Ports", March 2002.
   URL: http://www.dshield.org/topports.html

5. Microsoft Corporation, " IIS 5.0 Baseline Security Checklist ", 2001.
   URL: http://www.microsoft.com/technet/treeview/default.asp
   ?url=/technet/security/tools/iis5cl.asp

6. SANS Institute, Information Security Reading Room, " Securing Microsoft's Internet Information Server 5.0 ", Ben White, August 31st 2001.
   URL: http://rr.sans.org/web/sec_IIS.php

7. World Wide Web Consortium (W3C), "RFC2616, Hypertext Transfer Protocol -- HTTP/1.1", June 1999.
   URL: http://www.w3.org/Protocols/rfc2616/rfc2616.html

8. The Internet Engineering Task Force (IETF), "RFC 0791, Internet Protocol", September 1981.
   URL: http://www.ietf.org/rfc/rfc0791.txt

   The Internet Engineering Task Force (IETF), "RFC 0793, Transmission Control Protocol", September 1981.
   URL: http://www.ietf.org/rfc/rfc0793.txt

9. Incidents.org, "Incidents.org – By the SANS Institute", 2002.
   URL: http://www.incidents.org

10. SecurityFocus, "SecurityFocus – ARIS Threat Management System", 2002.
    URL: http://www.securityfocus.com/corporate/products/aris.shtml

12. eEye Digital Security, " .ida "Code Red" Worm ", July 17, 2001.
    URL: http://www.eeye.com/html/Research/Advisories/AL20010717.html

15. Snort.org, " Snort Users Manual – Snort Release: 1.8.5", 2002.
    URL: http://www.snort.org/docs/writing_rules/

18. Squid-cache.org, "Squid reverse proxy support", 2001.
    URL: http://devel.squid-cache.org/rproxy/

Software

3. Microsoft Corporation, "Security Tools", 2002.
URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tool/tools.asp

11. eEye Digital Security, "SecureIIS, Application Firewall"
URL: http://www.eeye.com/html/Products/SecureIIS/index.html

13. Microsoft Corporation, "URLScan Security Tool"
URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/tools/urlscan.asp

14. Snort.org, "Snort"
URL: http://www.snort.org

16. Volera, "Excelerator"
URL: http://www.volera.com/products/acceleration/excelerator.html

17. Squid-cache.org, "Squid Web Proxy Cache"
URL: http://www.squid-cache.org/