OWASP
The Open Web Application Security Project
6th OWASP AppSec Conference - Italy 2007

"ModSecurity Core Rule Set": An Open Source Rule Set for Generic Detection of Attacks against Web Applications

Ofer Shezaf, ModSecurity Core Rule Set Project Leader, OWASP Israel Chapter leader, CTO, Breach Security

Breach Security, 11 Bareket St, Herzelia, Israel, 46733
ofers@breach.com

Abstract: Misused based (i.e rules or signature based) attack **detection** is usually associated with protection from exploit of known vulnerabilities. However, most web applications are custom made and their vulnerabilities might be discovered by the attacker first. As a result web application protection usually employs positive model misuse detection or anomaly based detection. However, unlike negative security rules, such techniques are either very complex or require a large amount of manual configuration.

The ModSecurity Core Rule Set is an open source rule set aiming at providing effective protection using misuse based negative security model for web applications.

This paper will discuss the advantages, as well as the challenges and limitations of ModSecurity Core Rule Set as a negative security only rule set and as an open source project used by many but also scrutinized by many.

Keywords: Web Application Firewall, WAF, Intrusion Detection, Web Application Security, ModSecurity, Open Source.

1. Introduction

Web Application Firewalls (WAF) are a new breed of information security solutions designed to protect web applications from attacks. WAF solutions are capable of preventing attacks that network firewalls and intrusion detection systems can't. They also do not require modification of the application source code.

The Web Application Security Consortium's Web Application Firewall Criteria [11] is the most comprehensive definition to what is and what is not a web application firewall.

ModSecurity [2] is an open source application firewall under GPL [3]. It is probably the most widely installed WAF with more than 10,000 installations [4]. It boasts a robust rules language and is many times used as a reference implementation for web application firewalls.

The Core Rule Set, bundled with ModSecurity is a set of ModSecurity rules that implement a negative security model for protecting application firewalls. The Core Rule Set does not possess any knowledge on the protected application and therefore is a generic Rule Set. This document discussed how a generic rule set can protect applications using the Core Rule Set as an example.

2. Web Application Firewall Protection Strategies

Most web applications are custom programmed either internally at organizations or by a contractor. As a result signature based protection as commonly used by intrusion detection and prevention systems (IDPS), which relies on detecting known vulnerabilities, provides only limited security to web applications.

Instead, application firewalls enable the use of the following strategies in order to provide protection to web applications.

2.1 External patching

External patching, also known as "just-in-time patching" or "virtual patching" is a limited protection method that has importance due to the common software lifecycle management process in organizations. The later a vulnerability is found in the development and deployment process the price of fixing it gets higher as it disrupts the development process and delays deployment. If the vulnerability is found after an application is deployed the problem is bigger as taking the application off line may be impossible, forcing the organization to leave the application operative and incur the risk.

A WAF can be used to provide protection from a specific vulnerability without modifying the application. To do that, a rule is created and implemented in the WAF that provides additional validation to the specific field vulnerable.

For example, if the user name field in an application is vulnerable to SQL injection [5] attack, and the usernames are alphanumeric in the specific application, the following rule, in ModSecurity rules language, would provide an external patch:

```
<LocationMatch "^/login.php$">
            SecRule ARGS:username "!^\w+$" "deny,log"
</LocationMatch>
```

The rules checks that the "username" field on the "/login.php" page accepts only alphanumeric characters. Otherwise the request is denied and an alert is issued.

2.2 Positive security model

Positive security model is a comprehensive security mechanism that provides an independent input validation envelope to an application. By defining rules for every parameter in every page in the application the application is protected by an additional security envelop independent from its code. For example, the following rules in ModSecurity rule language provide such comprehensive security to Exchange Outlook Web Access login page:

```
SecDefaultAction "log,deny,phase:2"
<LocationMatch "^/exchweb/bin/auth/owaauth.dll$">
SecRule REQUEST_METHOD !POST
SecRuleARGS:destination"!^….$" (full regular expression left
out for simplicity)
SecRule ARGS:flags "[0-9]{1,2}"
SecRule ARGS:username "[0-9a-zA-Z]{,256}"
SecRule ARGS:password ".{,256}"
SecRule ARGS:SubmitCreds "!Log.On"
SecRule ARGS:trusted "!(0|4)"
</LocationMatch>
```

Learning

The limitation of this model is that it requires deep knowledge of the application and a considerable on going effort to maintain the rule set. The maintainer needs to define such rules for each parameter and page in the application. Essentially the rules have to follow closely the application and every change in the application requires a modification to the rule set as well.

In order to reduce the effort required, different learning mechanisms have been implemented. In a session based learning approach [6] rules are dynamically created based on previous transactions in the session. Specifically, based on forms returned by the web server the WAF generates validation rules for input submitted using this form by a user, validating field existence, length and special attributes such as "hidden" or "read-only". Other dynamic validation rules include rules limiting the allowed URLs only to those appearing in links in previous pages and rules limiting cookie values. However, this approach is limited since the information sent by the server does not convey all the information required to generate a rule. For example, the type of a parameter is not available. Furthermore, this method became nearly obsolete with the major shift to client side scripting, interactive web front ends such as AJAX [find] and web services. In all these technologies the client sends requests that are not based on previous server responses or are hard to determine from the responses.

More recently anomaly based learning approach [7] was suggested. In anomaly based approach an input validation profile is created for an application based on observing real usage traffic and determining normal usage patterns. As with most anomaly based detection techniques, the key challenges are differentiating between attacks and non malicious abnormal traffic, not including in the normal usage profile information derived from attacks and compensating for time based variability in the usage profile.

Differentiating attack traffic from non malicious abnormal traffic is a major challenge for monitoring systems, but is less severe for protection only systems as many non attack abnormal requests can be blocked as they would not generate useful results. The problem can be further reduced by using a detected anomaly only as an indicator and determining an attack only based on multiple indicators, both anomaly based and other.

2.3 Negative security model

A negative security model (or misuse based detection) is based on a set of rules that detect attacks rather than allow only valid traffic. The Core Rule Set discussed in this document is a negative security model rule set.

It is important to note that the differentiation between negative and positive security models is subjective and reflects how tight the security envelope around the application is. A good example would be limiting the characters allowed in an input. Since the character set is a closed set, providing a white list of permitted characters is actually similar to providing a black list of forbidden characters including the characters complementing the 1st group.

A negative security model is very common to Intrusion Detection and Prevention systems (IDPS). Therefore it is very important to understand what the differences are between the negative security model provided by an IDPS and the negative security model provided by a WAF.

Robust HTTP and HTML parsing

A WAF employs an HTTP and HTML parser to analyze the input stream. The parser should be able to understand specific protocol features including content encoding such as chunked encoding or multipart/form-data encoding, request and response compression and even XML payload.

In addition the parser must be as flexible as the environment protected as many headers and protocol elements are not used according to RFC [8,9] requirements. For example, while the RFC requires a single space between the method and the URI in the HTTP request line, Apache allows any sequence of whitespace between them. Another example is PHP unique use of parameters: in PHP leading and trailing spaces are removed from parameter names.

In a proxy deployment a stricter parsing may be acceptable, but if the WAF is deployed in any way in which only a copy of the data inspected, the WAF has to be at least as flexible as the web server in order to prevent evasion. IDPS systems that fail to do so can be easily evaded by attackers.

Protocol Analysis

Based on the parsed info, a WAF must break up the HTTP stream into logical entities that can be inspected, such as headers, parameters and uploaded files. Each element will be inspected separately not just for its content, but also for its length and count.

In addition a WAF must correctly divide the network stream when keep-alive HTTP connections are used to unique request and replies, and correctly match requests and replies.

In proxy deployment, a WAF or IDPS failing to match correct request and response data may be attacked using HTTP response splitting [10] attacks or HTTP request smuggling attacks [11]. In a passive out of line mode, the WAF or IDPS may be easily circumvented.

Anti Evasion features:

Modern protocols such as HTTP and HTML allow the same information to be presented in multiple ways. As a result signature based detection of attacks must inspect the attack vector in any form it may be in. Attackers evade detection systems by using a less common presentation of the attack vector. Some common evasion techniques include using different character encodings for the attack vector or using none canonized path names. In order to prevent evasion a WAF would transform the request to some normalized form before inspection.

While modern IDPS systems may support anti-evasion techniques, those are limited to well defined parts of the request such as the URI. A WAF can selectively employ normalization functions for different input fields for each inspection performed. For example, while an IDPS would normalize the URI, a WAF can normalize an HTML form field that accepts path names as input.

Rules Instead of Signatures

As the tests employed by a WAF are more complex, it cannot rely solely on signatures and requires a more robust rules language to define the tests. For example, the following features exist in ModSecurity rules language [12]:

- Operators and logical expressions – a WAF can check an input field for attributed other than its content, such as its size or character distribution. Additionally a WAF can combine such atoms to create more complex conditions using logical operators. For example, a WAF may inspect if a

field length is too long only for a specific value of another field, or alternatively check if two different fields are empty.

- Selectable anti-evasion transformation functions – as discussed above, each rule can employ specific transformation function.

- Variables, sessions & state management – since the protocols inspected keep state, the rules language needs to include variables. Such variable can persist for a single transaction, for the life of a session, or globally. Using such variables enables the WAF to aggregate information and therefore detect an attack based on multiple indications during the life span of transaction or a session. Attacks that require such mechanisms to detect are brute force attacks and application layer denial of service attacks.

- Control structures – a WAF rules language may include control structures such as conditional execution. Such structures enable the WAF to perform different rules based on transaction content. For example, if the transaction payload is XML, an entirely different set of rules may be used.

3. ModSecurity Core Rule Set

The ModSecurity Core Rule Set is a negative security rule set for web application firewalls distributed as an open source project under GPL. It is probably the only comprehensive rule set available in an open source form or its type and therefore provides an important opportunity to examine the effectiveness of a negative security model for web application protection.

The core rule set is available at http://www.modsecurity.org/projects/rules/index.html

While the Core Rule Set may be translatable to other web application firewalls, it does draw a lot of its power from ModSecurity. Specifically, ModSecurity robust rules language, granular parsing and wealth of anti-evasion transformation functions are used by the Core Rule Set.

The benefits and challenges of being Open Source

A negative security rule set has some major advantages and disadvantages from being an open source project. On the positive side, it is heavily tested both for false positives and false negatives by a large open source user base. Actually each such report, once solved, is added to the rule set regression test so that future changes would not create the same false alert. This is especially important due to the difference between the RFC specifications of the different protocols and the real world implementation.

On the other hand an open source rule set can be analyzed by hackers in order to find vulnerabilities. So just like open source software, vulnerabilities are more often found in open rule sets. As a result, only a well maintained open source project can benefit from the security advantages of being an open source project. As the core rule set is backed by a commercial company it enjoys this benefit.

4. Application Layer Signatures

The most important security mechanism employed by the Core Rule Set is signatures, just like its IDPS counterparts. In this section we will examine the difference between IDPS signatures and the Core Rule Set signatures.

4.1 Case study: or 1=1

The attack vector or 1=1 [13] is a classic example of an SQL injection attack vector. It is used to force SQL queries to return true value. For example, this attack vector is used to bypass login forms even when user name and password are not known. As a result, it is often used as a signature in IDPS systems to discover SQL injection.

Unfortunately, this signature can be easily evaded, using any of the following attack vectors[14]:

- or 1%3D1 – in this case the equal sign is encoded using URL encoding.
- or%091 =1 – the expression accepts any number of spaces or any other white space character such as horizontal tab between any of its tokens.
- or /**/ 1 /* this is a comment */ = 1 – inline comments (supported by MySQL can be added between any of the tokens to evade IDPS.

All these vectors can be detected using a robust regular expression and proper transformation functions, for example /or\s+1=\s+1/Ui, using Snort regular expression syntax in which the U modifier implied URI decoding and path normalization. In this case the transformation functions should also include a function to remove the comments. Alternatively the regular expression can be enhanced to remove comments.

But an attacker does not need to use the above expression at all. As the goal of the injection is to produce a true value, any expression can be sufficient including 2>1. Actually the injection true value does no have to be an expression: any value interpreted as true will work, including constants. Different databases accept different values as constants. MS-Access for example will accept 1, "1" and "a89" as true values so the injection string can simple be or 'a89'.

If the injection requires a string, the combination of the keyword or and the single quote might be sufficient as signature: or\s*' but this is signature is prone to false positives and would not work in cases where the injected true value can be a number. As we can see there is no simple, catch all generic signature to detect such attacks.

4.2 Reference IDPS signatures

In order to analyze generic application layer signatures, we need to be familiar with standard network based signatures. A case study is "bleeding edge" snort signature for Bugtraq vulnerability #21799. This vulnerability in the Cacti open source graphing software was picked quite at random.

The exploit references on Bugtraq vulnerabilities archive is [15]:

```
/cacti/cmd.php?1+1111)/**/UNION/**/SELECT/**/2,0,1,1,127.0.0
.1,null,1,null,null,161,500, proc,null,1,300,0, ls -la >
./rra/suntzu.log,null,null/**/FROM/**/host/*+11111
```

And the signature is [16]:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(
    msg:"BLEEDING-EDGE WEB Cacti cmd.php Remote Arbitrary
SQL Command Execution Attempt"; flow:to_server,established;
    uricontent:"/cmd.php?"; nocase;
    uricontent:"UNION"; nocase;
    uricontent:"SELECT"; nocase;
    reference:cve,CVE-2006-6799; reference:bugtraq,21799;
    classtype: web-application-attack; sid:2003334; rev:1;
)
```

While snort has some anti-evasion techniques such as case insensitivity and URI decoding, this signature still falls short of detecting an exploit of the vulnerability. It is gears only towards detecting the specific attack vector shown above. Any other exploit such as blind SQL injection would not be detected. It also searches for the keywords only in the request line, while many development environments would allow for parameters to be provided both in the POST and GET payload. Additionally this signature is prone to false positives as both select and union are common English words and since the signature do not require any word delimiters the signature will also be satisfied by the words "Selection" and "Reunion". In many cases such a signature has to be turned off.

4.3 Application Layer Signatures characteristics

A generic application layer signature is still based on detecting text patterns in the payload inspected. Unfortunately, as shown earlier, most patterns are not distinctive enough and may cause false positives:

- xp_cmdshell – this MS-SQL function is a strong enough indicator of SQL injection by itself.

- "<", single quote – Some characters are very indicative of attacks such as SQL injection or XSS, but are perfectly acceptable in free text, to quote, or as an apostrophe.

- select, union – are examples of SQL keywords that are also valid English words.

In order to achieve precise attack detection using such patterns, we need to aggregate multiple indicators. Some aggregation functions that can be used are:

- A very strong indicator by itself - xp_cmdshell, varchar.

- A sequence of indicators: union …. select, select … top … 1.

- Amount of indicators: script, cookie and document appearing in the same input field, with no order or distance limitations.

- A sequence of indicators over multiple requests from the same source.

4.4 A Generic Signature example

The following is the Core Rule Set signature for detecting SQL injection, for clarity it is abbreviated. The regular expression used is complex due to optimization done to enhance regular expression matching performance. The signature employes different techniques in order to detect more attacks:

- A large number of SQL injection patterns are search for. These patterns include both very strong keywords as well as sequences of weaker SQL keywords.

- All patterns use word boundaries, if applicable, to reduce false positives.

- The patterns are searched in all input fields that are not known to generate false positives, including POST and GET parameters and HTTP headers.

- Transformation function are used to ensure that no encoding or SQL comments evasion technique can bypass the signature match.

```
SecRule REQUEST_FILENAME|ARGS|ARGS_NAMES|
REQUEST_HEADERS|!REQUEST_HEADERS:Referer \

            "(?:\b(?:(?:s(?:elect\b(?:.{1,100}?\b(?:(?:length
|count|top)\b.{1,100}?\bfrom|from\b.{1,100}?\bwhere)|.*?\b(?
```

```
:d(?:ump\b.*\bfrom|ata_type)|(?:to_(?:numbe|cha)|inst)r))|p_
(?:(?:addextendedpro|sqlexe)c|(?:oacreat|prepar)e|execute(?:
sql)?|makewebtask)|ql_(?:… … … \

           “capture,log,deny,t:replaceComments,
t:urlDecodeUni, t:htmlEntityDecode, t:lowercase,msg:'SQL
Injection Attack. Matched signature
<%{TX.0}>',id:'950001',severity:'2'“
```

4.5 Specific application layer signature

Application layer signatures are better than simple signatures also when applied to a specific vulnerability. As noted above such a security scheme is called virtual patching. A virtual patch for Bugrtaq vulnerability #21799 would be:

```
<LocationMatch :"/cmd.php$">
    SecRule QUERY_STRING "^[\d\s]*$” “deny,log"
</LocationMatch>
```

Such a patch addresses the actual vulnerability, by limiting the query string to digits and spaces only, as intended by the program. It is much simpler and handles well most evasion techniques. However, the limitation of the method is that it can be employed only to a server running the vulnerable application as other applications that have a page called "cmd.php" may break if this rule is applied.

5. Protocol Validation

Apart from signatures, the Core Rule Set also validates the HTTP syntax as used in the request and reply. Many generic web application vulnerabilities such as Response Splitting, Request Smuggling and Premature URL ending [17] rely on inaccurate analysis of HTTP by the server. This is also true for specific vulnerabilities [18].

5.1 RFC & Real World protocol compliance

It is important to note that the Core Rule Set does not check that the request and reply strictly adhere to the different HTTP RFCs. While this can be done using ModSecurity rules, it has two drawbacks: Many HTTP clients and servers do not follow the RFC and not every RFC violation has security implications making the test a waste or resources.

Some protocol validations performed by the Core Rule Set are:

- Ensure Content-length is different than 0 for and only for none GET/HEAD methods.

- Detect non ASCII characters or encoding in headers (excluding headers such as "referer" or "cookie" which may include such characters)
- Valid use of headers (for example, content length is numerical)

5.2 Policy Limitations

In addition some real world limitations can be applies to requests. Those limitations are not in violation of the protocol definitions but transactions that violate the limitations cannot be used in real world applications. These limitations can be expanded the more specific the environment protected is, blurring the line between negative and positive security model. For example, if the application protected was developed using PHP, extensions belonging to other development environments such as ASP can be rejected.

Such policy based limitations included in the Core Rule Set are:

- Require headers such as Host, Accept or User-Agent.
- Check that the Host header is not an IP address.
- Restrict use of HTTP methods such as CONNECT, TRACE or DEBUG, and if the application does not require them also WebDAV methods.
- Restrict URL file extensions.
- Restrict content types for both requests and replies.

6. Additional Detection Methods

Some additional protection methods are used by the Core Rule Set are listed in the following paragraphs.

Malicious Robots

The Core Rule Set detects malicious robots mainly based on information provided by the client so a smart attacker can evade the rules easily. The malicious robots detection rules are included in the Core Rule Set to reduce nuisance by eliminating wide spread unsophisticated attacks, which reduce the load on the server and reduce the alerts count. One of the annoying phenomena that these techniques fight is comments spam.

Among the techniques that can be used to block such malicious robots are:

- Detecting non browser attributes, such as requests missing a User-Agent, Host or accept header.

- Detecting non browser or known malicious User-Agent headers.

- Blocking a black list of IP addresses, known to be used by malicious robots.

Using the same techniques the Core Rule Set detect security scanners and other, positive robots such as search engines. As a side benefit, while doing that, the Core Rule Set also confuse security testing software such as HTTPrint.

Trojans and Viruses

Trojans and virus infected files are a major problem at hosting environments. In such environments uploading files is usually allowed, which enable hackers to use legal uploading mechanism to upload Web Trojans and exploit the system.

Web Trojans are usually web servers' scripts written in PHP or ASP. Once uploaded, they enable the hacker to execute system functions from a browser.

In order to detect Trojans and Viruses the Core Rule Set employs the following techniques:

- Inspecting uploaded files for Viruses using an external AV such as ClamAV. [19]

- Inspecting uploaded files for Trojans using signatures – this is important since most AV software packages do not detect Web Trojans.

- Detecting access to Trojans, either using known signatures of Trojan access (for example, the x_key header is used by many some Trojans to authenticate users) or by detecting generic output of system functions such as an operating system type file listings.

Error conditions

The last line of defense is to detect errors printed by different web development systems. These errors are very useful to hackers, and many of them are embedded in a normal response rather than in an official error page accompanies by a 500 status code. Therefore using signatures to detect and block the printing of such errors severely limits the opportunities presented to a hacker.

7. Future Plans

The Core Rule Set is a live and thriving project. Apart from constant bug fixes and false positive reduction, we plan to enhance it to protect from more and more attacks. The following enhancements are planned for the future:

- Session bases protection – by correlating between detected events in a single session, attacks such as brute force password guessing and denial of service can be detected and blocked.

- XML - Using ModSecurity basic XML parsing capabilities, the Core Rule Set will offer basic protection for XML payloads such as soap. Such protection will include initially schema validation and contextual search for the existing signatures in XML data.

8. Benchmarking the Core Rule Set

So how effective is the Core Rule Set in actually protecting web applications? This is a very difficult question to answer and deserves an additional paper. Benchmarking a rule set requires measuring two indicators: the rate of false positives, i.e. how many valid requests where blocked and the rate of false negatives, i.e. how many attacks where not blocked by the rule set.

False Positives

In order test the Core Rule Set we used capture files of traffic at approximately 30 unrelated web sites. We converted the capture files into requests and reply pairs and used a special testing script to play them through a ModSecurity reverse proxy application firewall running the Core Rule Set. The script ensured that each request would cause the matching reply to be sent back through the proxy.

The main goal of this process was to test and fix the Core Rule Set, and we did find out the after 6^{th} web sites the number of false positives dropped dramatically. However we learned that the majority of the remaining false positives stem from three sources:

- While normal users never generate protocol violations, automated programs such as a site monitoring or mirroring programs often violate the HTTP protocol. If a web site is using such systems, they would generate false positives. Such a false positive is usually resolved by creating an exception for the source IP of the violating system for the specific violated rule.

- Applications that allow HTML editing, such as blog or wiki systems, usually breaks XSS detection rules. This is not really a false positive as these capabilities are a security risk. A specific web site may be willing to endure such a risk and exceptions are required to allow the HTML editing in relevant pages.

- While user generated input usually does not generate false positives, programmatic input such as parameter names may generate false positives. The reason is that many attack vectors and therefore attack signatures are also programming fragments. Since programmatic input is constant, exceptions can be easily created for it.

Fortunately all these issues can be resolved by adding a small numbers of exceptions.

False Negatives

Since the main goal of the Core Rule Set is to detect unknown attacks rather than known attacks, it is very difficult if not impossible to measure the false negative rate.

We do know that the Core Rule Set detects real attacks in both commercial and open source installation. However, how many attacks does it miss? Naively one may think that using one of the available application security scanners can help to determine the rate of false negatives. The caveat of this approach is that the results depend on the application selected for testing and any rule set can be optimized to protect any known application tested by any specific scanner.

One approach to determining false negatives rate is to use a random application or applications for testing. This is left for a future research.

References

1. The Web Application Firewall Evaluation Criteria (WAFEC), Project of the Web Application Security Consortium (WASC), Version 1.0, January 14th 2006 http://www.webappsec.org/projects/wafec/,
2. ModSecurity, Version 2.1, released February 27, 2007, http://www.modsecurity.org
3. GPL, GNU General Public License, Version 2, June 1992, http://www.gnu.org/copyleft/gpl.html
4. ModSecurity's Web Application Firewall Leads In Deployment Numbers But Lags In Usability, The Forrester Wave™ Vendor Summary, Q2 2006, by Michael Gavin, http://www.forrester.com/Research/Document/Excerpt/0,7211,39714,00.html
5. SQL Injection Attacks by Example, Steve Friedl, Jan 13th 2005, http://www.unixwiz.net/techtips/sql-injection.html

6. AppShield: Next Generation Reverse Proxy for a Secure Web Environment, 2002, www.gradian.co.uk/Resource_Lib/Sanctum/Reverse%20Proxies.pdf

7. Anomaly Detection of Web-based Attacks, Christopher Kruegel & Giovanni Vigna, Reliable Software Group, University of California, Santa Barbara, October 2003, http://www.cs.ucsb.edu/~vigna/publications/2003_kruegel_vigna_ccs03.pdf

8. RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, 1999 http://www.w3.org/Protocols/rfc2616/rfc2616.html

9. RFC By Category, Hypertext Transfer Protocol http://www.faqs.org/rfcs/np.html#HTTP

10. “Divide and Conquer”, HTTP Response Splitting, Web Cache, Poisoning Attacks, and Related Topics,
Amit Klein, Director of Security and Research, Sanctum, Inc. , March, 2004
http:// www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

11. HTTP Request Smuggling, Chaim Linhart, Amit Klein, Ronen Heled And Steve Orrin, A Whitepaper From Watchfire
http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

12. ModSecurity Reference Manual, Version 2.1.0 / (February 23, 2007)
http://www.modsecurity.org/documentation/modsecurity-apache/2.1.0/modsecurity2-apache-reference.html

13. SQL injection Basic Tutorial,
http://www.governmentsecurity.org/articles/SQLinjectionBasicTutorial.php

14. SQL Injection Signatures Evasion, By Ofer Maor, Application Defense Center Manager and Amichai Shulman, Chief Technology Officer, Imperva, April 2004
http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html

15. Cacti CMD.PHP Remote Command Execution Vulnerability
http://www.securityfocus.com/bid/21799/exploit

16. Bleeding Edge Signatures for snort
http://www.bleedingsnort.com/bleeding-web.rules

17. IDS Evasion Techniques and Tactics, Kevin Timm, May 7, 2002
http://www.securityfocus.com/infocus/1577

18. Apache HTTP Server Arbitrary HTTP Request Headers Security Weakness
http://www.securityfocus.com/bid/19661/info

19. http://www.clamav.net/