eval($WAF)$;

An evaluation of current web application firewall capabilities and techniques

FINAL PROJECT DOCUMENTATION

University of Applied Sciences Upper Austria
Campus Hagenberg, Secure Information Systems

L. Nothdurfter, W. Neudorfer, M. Kirchner

VERSION: 1.1; LAST CHANGE:
HAGENBERG, 4TH JULY 2009
Contents

1 Executive summary .................................................. 2
2 Project objectives ................................................... 3
3 Introduction into web application firewall technology and test setup .................................................. 5
4 Project results ......................................................... 7
  4.1 Implementation of test cases .................................. 7
  4.1.1 Vulnerability classes ........................................ 7
  4.1.2 Table of test cases ........................................... 9
  4.2 General description of evaluated web application firewalls .................................................. 16
  4.2.1 phion airlock ................................................. 16
  4.2.2 Artodefence Hyperguard ................................... 23
  4.2.3 F5 Networks BIG-IP ....................................... 31
  4.2.4 Breach Security ModSecurity ................................ 39
  4.3 Circumvention of default filtering mechanisms .................................................. 46
  4.4 Anomaly detection capabilities ................................ 50
  4.5 Test of parsing robustness using fuzzing techniques .................................................. 53
  4.5.1 Testing approach ............................................. 53
  4.5.2 Results ....................................................... 55
  4.6 Conduction of penetration tests ................................ 57
5 Conclusion ........................................................... 60

References ............................................................. 61
1 Executive summary

To protect an organisation from attacks on their IT infrastructure, perimeter firewalls are nowadays means of standard protective measures. Attacks on the application layer (e.g. web applications) cannot be effectively prevented by those systems as HTTP and HTTPS requests usually pass the firewalls mechanisms unfiltered and are forwarded directly to the web server. Web application firewalls therefore operate on a higher network layer seeking to prevent application level attacks by analysing the user data transmitted via HTTP or HTTPS.

By filtering requests and responses of the web server, the exploitation of vulnerabilities in web applications and the leakage of sensitive data should be prevented. However, the usage of web application firewalls cannot provide effective protection for all typically encountered vulnerability classes or bogus web server configuration issues.

The project at hand evaluated current web application firewall capabilities and techniques to state in which scenarios and for which vulnerability classes the usage of these products can be recommended. On the other side the drawbacks of the current products and techniques have been found and demonstrated.
2 Project objectives

The project at hand tested current web application firewall technology as far as their effectiveness and the coverage of vulnerability classes are concerned. For the test cases a series of vulnerable applications has been implemented. The security problems of the vulnerable system not only include susceptible applications but also vulnerable services that are needed to operate web applications (e.g. a web server). The vulnerable web applications are of both types, proprietary developments and publicly available software systems at a vulnerable patch level or version. The proprietary developments of web applications are represented by mock implementations (small applications that do not provide much or any functionality but are vulnerable to certain attacks). To classify all implemented vulnerabilities, certain vulnerability classes (for example „path traversal“ or „file inclusion“) have been defined and are explained later in greater detail.

To be able to test web application firewall technology, test equipment of product vendors has been obtained. The equipment was used to get an impression of the current technical standard by using it for test cases in connection with the previously implemented vulnerable applications. The following vendors or community products have been used throughout the project:

- phion airlock
- Artofdefence Hyperguard
- Breach Security ModSecurity
- F5 Networks BIG-IP ASM

The first part of the project dealt with an evaluation to determine which functions current web application firewalls provide to enhance the security level of the protected web server or web application. It was analysed in which scenarios the deployment of web application firewall techniques can be recommended and which vulnerability classes can effectively be prevented by those systems. The vulnerability classes defined during the implementation of the test cases have been used to determine for which classes the current technology only provides inadequate protection and where the protection is satisfactory.

The following list covers all project objectives the project team worked on:

- To be able to conduct testcases in connection with web application firewall technology and products, a series of vulnerable applications is implemented. The security issues of the vulnerable system not only include susceptible applications but also vulnerable services that are needed to operate web applications (e.g. misconfiguration of the web server). The vulnerable web applications are of both types: proprietary developments as well as publicly available software systems at a vulnerable patch level. To classify all implemented vulnerabilities, certain vulnerability classes (for example „path traversal“ or „file inclusion“) are defined.

- To be able to test web application firewall technology, test equipment of accordant product vendors is obtained. The equipment is used to get an impression of the current technical standard by using it for testcases in connection with the previously implemented vulnerable applications.
• It is evaluated which vulnerabilities of the previously set up applications the web application firewalls cover by default, which vulnerabilities require additional configuration and which vulnerability classes can hardly be covered.

• It is analysed in which scenarios the deployment of web application firewall techniques can be recommended and which vulnerability classes can effectively be prevented by those systems.

• It is evaluated which functions of web application firewalls only provide inadequate protection and which vulnerability classes are left totally uncovered by the systems.

• Penetration test against the management interfaces of the web application firewall products (e.g. web interfaces or proprietary administration tools) are conducted.

• Based on the filtering results (which testcases have been blocked by which web application firewall) the project team seeks to circumvent the filter mechanisms so that vulnerabilities can be exploited regardless of the implemented filtering functions.

• The capabilities of current web application firewalls are evaluated as far as intrusion detection technologies are concerned (analysis of the number of requests, detection of anomalies, etc.).

• As web application firewall systems often have the possibility to (or need to) be adapted to the applications to be protected, it is evaluated which possibilities there are to „virtually patch“ known vulnerabilities in a web application that are not covered by the default ruleset are evaluated.

• As web application firewalls parse HTTP requests it is vital that these routines are implemented robustly and also can handle requests and responses that do not comply with the HTTP specification. Therefore these routines are tested for example by using fuzzing techniques.


3 Introduction into web application firewall technology and test setup

Web application firewalls operate as proxies between clients and web servers. All requests and optionally responses are checked for patterns that indicate attacks against the provided web applications. The matching parts of the requests are either deleted or the request is not forwarded at all. If a request is denied an appropriate error is risen and delivered as a HTTP response to the clients browser. The proxy can either be realized as a reverse or as a transparent proxy. In the case of a reverse proxy SSL connections have to be terminated on the web application firewall in order to be able to parse and check requests in clear text. In the case of a transparent proxy the SSL keys have to be stored at the web server and additionally on the transparent proxy. The web application firewall is subsequently able to read and decrypt the SSL handshake between the web server and its clients, extract the session keys and subsequently check all transmitted data.

As far as outbound filtering is concerned, web application firewalls can filter response pages from the server to the client. This for example enables filtering of credit card data, SQL errors or stack traces in general, as this information should not be transmitted to users.

As an additional function web application firewalls can be combined with load balancers or caching proxies. The main target of web application firewalls are applications that customers cannot change within a short period of time (for example because the source code or part of it is not available). If a vulnerability emerges it can then be virtually patched using the web application firewall. It can however be argued that web application firewalls are also of use with applications an organisation has full access to. In this case the web application firewall fulfills preventive measures and can be used in the time between the emerging of a vulnerability and the final fix in form of a new release (including the time for analysis, coding, testing and deployment).

Commonly known functions of web application firewalls are URL, cookie and form field encryption. All URLs, cookies and form field values that are transmitted from the web server to the client are intercepted and replaced by encrypted values. If for example the user clicks on a link, the client sends a request with an encrypted parameter. As the encrypted parameter would not be understood by the web server, the web application firewall transparently decrypts the values again and sends the request to the web server in clear text. Beside protection against parameter manipulation this can also be used against forceful browsing. As the web application firewall learns all links in a web page that are delivered from the web server to the client, the firewalls subsequently only allows access to those sites and functions it previously learned.

A second major function of web application firewalls is filtering based on regular expressions. Regular expressions (either predefined by the vendor or manually added by an administrator) can be applied on the whole web application or a part of it (beginning from a subdirectory structure). All request contents are checked against those regular expressions that describe common attack patterns. Examples are regular expressions for common SQL syntax (to prevent SQL injections) or JavaScript syntax (to prevent cross-site scripting attacks).

Web application firewall vendors describe that their products are also be meant to establish compliance with the Payment Card Industry Data Security Standard (PCI DSS)\(^1\). The goal of the standard is to enhance payment account data security, especially with credit card

\(^1\)For further information see https://www.pcisecuritystandards.org/.
transactions. Compliance is for example achieved by outbound filtering of credit card data as mentioned above.

An additional function that vendors currently start to integrate is a learning mode for subsequent anomaly detection (and intrusion prevention). In the first step the web application firewall does not block any requests but only analyses all requests and learns traffic patterns typical for the web application to be protected. When enough data has been collected, the learned patterns are used to generate a positive ruleset allowing all learned traffic and blocking everything else. It has to be considered that functions like this possibly lead into denial of service scenarios when legitimate traffic has not occurred during the learning phase and is blocked afterwards. It also has to be considered that after every change of the web server content or applications a new learning procedure has to be started to update the according internal data structures.

While web application firewall technology is currently evolving, some security researchers already consider web application firewalls effective and useful, as shown by the following quote:

"Alle getesteten Produkte bieten ein hohes Maß an Sicherheit gegen Angriffen auf Ebene der Applikationsschicht." [6]

To get a practical impression of current web application firewall technology and reach the projects objectives, a practical test scenario has been developed. Figure 1 shows the web server where the vulnerable images are hosted as well as the web application firewalls that operate as reverse proxies.

The web server hosts several vulnerable web applications (described in detail in chapter 4.1.2) that should be protected by the web application firewalls. To make all tests transparent and easily repeatable, a perl script has been developed that automates all tests and checks whether the applications are still vulnerable. All test cases have also been repeated once by hand to be able to analyse the reaction of the web application firewalls to the requests and to ensure the proper working of the test script.

All machines shown in figure 1 have been virtualized using VMware, except for the testing clients and the F5 Networks BIG-IP machine, which is provided as a hardware appliance from Triple AcceSSS IT. All other web application firewalls shown are distributed as software modules that have to be installed as reverse proxy machines (or alternatively directly on the web server).

---

2English translation: "All tested products offer an high level of security against application layer attacks."
3http://www.3xs-it.com
4 Project results

The following chapter covers the results as far as the project objectives and additional experience in the field of web application firewalls are concerned.

4.1 Implementation of test cases

To evaluate the capabilities of web application firewalls currently used in the field, a series of test cases has been developed. Every test case is assigned an according vulnerability class, an ID, a describing text covering the details of the vulnerability and the service type.

The ID indicates a unique number for differentiation purposes during the testing efforts. The description explains the vulnerability and how it can be exploited by an attacker or used to gain further information or access.

4.1.1 Vulnerability classes

The vulnerability class groups similar vulnerabilities in order to be able to state the effectiveness of current web application firewall technology as far as certain vulnerability classes are concerned. The following list gives an overview of all vulnerability classes covered by the project:

1. Information disclosure. This class summarises all vulnerabilities that enable an attacker to gain further information about a web application or its environment. The information is not needed to be disclosed to attackers and may enable them to start more
precise attacks or exploit other vulnerabilities in connection with the gained knowledge. Examples for information disclosure vulnerabilities are the dump of version information or directory listings.

2. **Brute force.** Vulnerabilities of this class enable attackers to conduct brute force attacks and may gain further privileges at the affected application. An example for this vulnerability class is the possibility to conduct unhindered brute force attacks against web-based login forms.

3. **Logical errors.** Logical errors concern vulnerabilities at the application level that may enable attackers to read, write or delete information that they normally should not have access to. A vulnerability of this class would for example allow an attacker to read private messages of other users by manipulating parameters and making use of an improperly implemented authorisation system.

4. **Violation of the minimal principle.** The minimal principle states that a user is only offered those functions, resources and permissions that are absolutely necessary to fulfill a certain task. A violation of the minimal principle is the access to administrative interfaces for ordinary users or publicly accessible example applications of application servers. A violation of the minimal principle may also result in a full server compromise, for example when a user can upload arbitrary files (e.g. PHP files) where only certain file types should be allowed.

5. **Path traversal and inclusion of external content.** Vulnerabilities of this class enable an attacker to include external content to a web application that was not intended by the developers. This can for example be achieved by manipulating parameters of a URL to include remotely or locally stored files or to influence the contents of displayed frames. Path traversal vulnerabilities enable an attacker to access information outside the source directories intended by the developers. Vulnerabilities of this class may also enable attackers to access information outside of the webroot and therefore can disclose sensitive system configuration files.

6. **Command injection.** Vulnerabilities that enable an attacker to execute arbitrary commands on the affected system are summarised to this vulnerability class. Vulnerabilities of this type arise when user input is used as parameters for locally executed programs.

7. **Interpreter injection.** Interpreter injection vulnerabilities include all vulnerabilities that enable an attacker to influence the evaluation routines of interpreter mechanisms. Common examples are SQL injections.

8. **Weak state management.** As HTTP is a stateless protocol web applications have to implement own mechanisms to keep state during consecutive requests. An example for a vulnerability of this class is a mechanism that generates weak session IDs that are either predictable or can be bruteforced and thus enabling an attacker to impersonate other users.

9. **Browser-based attacks.** Browser-based attacks either enable an attacker to directly engage ordinary users or need the unaware assistance of a legitimate user. Examples for this vulnerability class are cross-site scripting, cross-site request forgery or session fixation attacks.
The service type of a test case classifies vulnerabilities in the two types *basic* and *application-specific*. Whereas the *basic* type covers vulnerabilities in services that are essential for the proper functionality of web applications (e.g. the web server), the *application-specific* type describes vulnerabilities that result directly from improper implementation or configuration of web applications.

### 4.1.2 Table of test cases

The following table covers all test cases used during the project to evaluate the capabilities of web application firewalls currently used in the field.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Service type</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td><strong>Vulnerability class: Information disclosure</strong></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>The Apache web server in its default configuration offers a directory listing in the <em>icons</em>-folder. At the bottom of the listing detailed information about the server version, operation system, modules and module versions as well as the hostname or IP-Adress is disclosed.</td>
<td>Basic</td>
</tr>
<tr>
<td>102</td>
<td>The Apache web server in its default configuration discloses its version, host operating system as well as installed modules and their versions in the HTTP response header.</td>
<td>Basic</td>
</tr>
<tr>
<td>103</td>
<td>The file <em>index.html</em> in the folders <em>mysqlinjection_get</em> and <em>mysqlinjection_post</em> discloses the existence of a file that is referenced nowhere else. The accordant link is only commented out in the HTML source code and not removed.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>104</td>
<td>The directory <em>errormsgs</em> offers several scripts which rise either PHP database or PHP syntax errors that disclose information about absolute paths and database credentials.</td>
<td>Basic</td>
</tr>
<tr>
<td>105</td>
<td>The web server offers a file called <em>phpinfo.php</em> in its root directory which discloses a great number of sensitive information about the PHP installation as well as the web server configuration. The information includes the detailed PHP version information (which can subsequently be used to search for belonging known vulnerabilities), absolute web server paths (useful for file inclusion attacks), installed modules as well as configuration options in place.</td>
<td>Basic</td>
</tr>
<tr>
<td>106</td>
<td>The wordpress installation in the folder <em>wordpress</em> is susceptible to SQL injection attacks. If an incorrect SQL syntax is provided, a MySQL warning is raised disclosing information about the database layout.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>107</td>
<td>The directory <code>tempfiles</code> offers a script called <code>backup.php</code>. Additionally files like <code>backup.php.bak</code> and <code>backup.php.tmp</code> exist as they are created when developers make backups before they change source code files. Due to the changed file extension the PHP code is not interpreted any more by the web server and thus an attacker is able to download the source code of the script file.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>108</td>
<td>A weblog installation available under the directory <code>wordpress</code>. The login interface for the application is directly linked from the start page. After an invalid login attempt the application reveals whether the entered username or the provided password was wrong. This approach facilitates brute force attacks as first valid usernames can be searched which are then used for password brute force attempts.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>109</td>
<td>The scripts in the folders <code>userenumeration_get</code> and <code>userenumeration_post</code> enable a user to login. If the login attempt fails, the application tells the user whether the entered username or password was false. This facilitates the process of finding valid usernames and performing brute force attacks.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>110</td>
<td>The web server provides several private directories that are not linked from any web pages and contain several legacy files like backups or temporary files that are not necessarily needed. As these files are not needed for productive functions this is considered a violation of the minimal principle. The following directories are part of this test case: <code>admin</code>, <code>administrator</code>, <code>backup</code>, <code>privat</code> and <code>private</code>.</td>
<td>Application-specific</td>
</tr>
</tbody>
</table>

**200  Vulnerability class: Brute force**

| 201 | The folders `bruteforce_get` and `bruteforce_post` offer a user the possibility to login. The user `admin` has chosen a weak password. As the password is short and does not contain and special characters, it can easily be brute forced by an attacker. | Application-specific |

**300  Vulnerability class: Logical errors**
<table>
<thead>
<tr>
<th>301</th>
<th>The folder <em>imageapp</em> offers registered users the possibility to upload and share photos. After the login procedure the user sees all currently available photos either uploaded by him- or herself or by other users of the same application. The user can manage the photobase by uploading new images or by deleting existing ones. All users underlie the restriction that they can only delete images they have uploaded themselves and not the images that are provided by other users. An implementation flaw in the file <code>delete.php</code> however enables an authenticated user to delete arbitrary photos regardless of who uploaded them. Therefore the URL parameter <em>file</em> has to be changed to the filename of the image to be deleted.</th>
<th>Application-specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>302</td>
<td>The project management tool available in the directory <em>collabtive</em> requires a valid login to access administrative functions. The login can be bypassed by directly requesting the URL to administrative functions. The file <em>admin.php</em> can for example be called without prior authentication.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>303</td>
<td>The application under the folder <em>phpbb2</em> offers a bulletin board that requires user to login before posts can be read or written. By crafting cookie values the authentication mechanism can be bypassed.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>304</td>
<td>The scripts in the directory <em>recursive_output_get</em> and <em>recursive_output_post</em> replay a user-defined text a user-defined number of times. If the number chosen by the user is high enough and the script is called several times, this can lead into a denial of service scenario.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>305</td>
<td>The Jabber-Server <em>openfire</em> requires administrators to login before settings can be changed. Due to a logical error and a directory traversal vulnerability the authentication can be bypassed. The affected file is part of the setup routines and is called <em>log.jsp</em>.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>400</td>
<td><strong>Vulnerability class: Violation of the minimal principle</strong></td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>The standard configuration of the web server allows arbitrary users to conduct HTTP <em>trace</em> requests. The <em>trace</em> method is a debug functionality which echoes the content of the received HTTP request back to the user. If the HTTP request has been changed during transmission between the browser and the web server (for example by a reverse proxy or by filtering mechanisms), the user can take insight of that. This issue can possibly allow an attacker to gain knowledge of intermediary filtering mechanisms and enable him or her to explore the mechanisms in detail.</td>
<td>Basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>402</td>
<td>The host provides access to the <em>phpmyadmin</em> login interface without restriction of IP addresses or additional protection using <em>htaccess</em>. As access to this interface is not needed by regular users this is considered a violation of the minimal principle. Additionally, the software version of <em>phpmyadmin</em> is disclosed at the login page, which allows an attacker to easily search for known vulnerabilities of the respective version. Application-specific</td>
<td></td>
</tr>
<tr>
<td>403</td>
<td>The application in the folder <em>imageapp</em> allows users to upload and manage photos. The file <em>delete.php</em> which is normally used to remove uploaded files can also be used to delete arbitrary files of the web server the <em>www-data</em> has permissions for. This is done by changing the parameter <em>file</em> in the accordant URL and by making use of directory traversal techniques. Application-specific</td>
<td></td>
</tr>
<tr>
<td>404</td>
<td>The application in the folder <em>imageapp</em> offers registered users the possibility to upload images and share them with other users. The upload functionality does not check the file type of the uploaded files and therefore enables an upload of arbitrary files. An attacker may upload files with PHP extension that are interpreted by the server and enable a full server compromise. Application-specific</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td><strong>Vulnerability class: Path traversal and inclusion of external content</strong></td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>The scripts in the directory <em>rfi_get</em> and <em>rfi_post</em> have an implementation flaw that allows an attacker to include remote files that are also interpreted on the server. This allows an attacker to fully compromise the affected server. The affected parameter is called <em>module</em> in the file <em>index.php</em>. Application-specific</td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>The news management application under the directory <em>webnews</em> has an implementation flaw that allows an attacker to include arbitrary files. The affected script is called <em>parser.php</em>, the according parameter is <em>WN_BASEDIR</em>. To successfully exploit the vulnerability, the suffix of the inclusion must be truncated by using a null byte. Application-specific</td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>The scripts in the directories <em>pathtraversal_get</em> and <em>pathtraversal_post</em> have an implementation flaw that enable an attacker to traverse an URL parameter and include any locally available files. The affected parameter is called <em>include</em> in the file <em>index.php</em>. Application-specific</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td><strong>Vulnerability class: Command injection</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>601</td>
<td>The folders <em>commandexecution_get</em> and <em>commandexecution_post</em> offer an application that can be used to ping foreign hosts. As the application passes the provided IP address to a command line tool and the user input is not sanitized correctly, arbitrary commands can be inserted and executed on the server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application-specific</td>
<td></td>
</tr>
<tr>
<td>602</td>
<td>The contents of the form field provided in the folder <em>bufferoverflow</em> are passed to a CGI program implemented using the C programming language. During the internal handling of the provided data, the content is copied to a limited buffer. If the user-provided data is longer, an buffer overflow occurs and the server responds with an error code 500 (internal server error). The buffer overflow is also logged in the error log of the Apache error log as a premature end of script error. Given protective measures of modern operating systems as address space layout randomization and stack protection it is considered unlikely that an attacker may be able to execute malicious code. Exploitation would only be possible on older operation systems. Therefore an working exploit was not developed in the course of the project at hand. An attacker is nevertheless able to disrupt the execution of the affected service or to execute arbitrary code using brute force mechanisms to guess return addresses.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application-specific</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td><strong>Vulnerability class: Interpreter injection</strong></td>
<td></td>
</tr>
<tr>
<td>701</td>
<td>The application in the folders <em>commandinj_eval_get</em> and <em>commandinj_eval_post</em> accepts a parameter that is subsequently processed by a PHP script. The parameter is used in a PHP eval statement where the input data is not sanitized correctly. Therefore an attacker is able to inject arbitrary PHP code and also execute arbitrary commands within it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application-specific</td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>The folders <em>mysqlinjection_get</em> and <em>mysqlinjection_post</em> offer a login form which is susceptible to SQL injection attacks. The authentication mechanism can be bypassed by inserting SQL code that comments out the password validation mechanism.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application-specific</td>
<td></td>
</tr>
<tr>
<td>703</td>
<td>The version of the weblog software <em>wordpress</em> available in the same directory has a known SQL injection vulnerability. The parameter indicating the blog category (<em>cat</em>) can be used to alter the SQL statement that is used to perform database queries. An attacker therefore is able to read sensitive information like user data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application-specific</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td><strong>Vulnerability class: Weak state management</strong></td>
<td></td>
</tr>
<tr>
<td>801</td>
<td>The scripts in the folder <em>weaksessionid</em> requires a user to login. To take track of the user over multiple HTTP requests without the need to relogin for every request, a session ID mechanism was implemented. The server sends the session ID to the client using cookies and stores the associated user information in a database on the server. The implementation of the session ID mechanism is insufficient as the ID value consists only of two numerals and can therefore easily be bruteforced by an attacker to impersonate a legitimate user.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>802</td>
<td>After a successful login at the application in the folder <em>weaksessionid</em> a user is provided a list of information about his session ID, his user type, etc. There is also a section providing information about other users and their account details, but this section is only available for administrators and not for regular users. As the information whether a logged-in user is a regular user or an administrator is stored in cookies, an attacker can tamper the stored data and therefore access the data otherwise not shown.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>803</td>
<td>The login mechanism in the folder <em>sessionfixation</em> is vulnerable to session fixation attacks. An attacker is able to predefine the session ID value by setting the according URL parameter <em>sessionid</em> for the file <em>index.php</em>. If the attacker can convince a user to login using the crafted link, the account of the affected user can be overtaken.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>900</td>
<td><strong>Vulnerability class: Browser-based attacks</strong></td>
<td></td>
</tr>
<tr>
<td>901</td>
<td>The application in the folder <em>imageapp</em> allows users to upload photos and share them with other users. Photos can only be deleted by the user who uploaded them. As the script <em>delete.php</em> does not perform any checks of volatile parameters and only requires a valid authentication and the filename to be deleted, an attacker may delete arbitrary photos with the help of cross-site request forgery. Therefore a user must be persuaded to visit a manipulated website that injects the deletion command.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>902</td>
<td>The scripts in the folders <em>recursive_output_get</em> and <em>recursive_output_post</em> allow users to make inputs in form fields. In the further handling of the entered parameters, the contents are used for outputs to the resulting website without performing sanity checks. Therefore an attacker is able to perform cross-site scripting attacks and inject arbitrary script code into the resulting website.</td>
<td>Application-specific</td>
</tr>
<tr>
<td>903</td>
<td>The wordpress installation in the folder <em>wordpress</em> is susceptible to SQL injection attacks. When an incorrect SQL syntax is used, an appropriate error is risen replaying the user input without sanity checks. Therefore cross-site scripting attacks can be conducted.</td>
<td>Application-specific</td>
</tr>
</tbody>
</table>
The application in the folder `weaksessionid` requires a user to login and afterwards stores user information in cookies that are send to the client. User-provided cookie values are included in the resulting web page without sanity checks. Therefore cross-site scripting attacks can be conducted by changing cookie values.

Application-specific

The application in the folders `commandexecution_get` and `commandexecution_post` allow to ping hosts using a web frontend. The provided parameters are not sanitized correctly enabling an attacker to conduct cross-site scripting attacks.

Application-specific

The Jabber-Server `openfire` offers a login interface using the file `login.jsp`. The according script is affected by a cross-site scripting vulnerability using the parameter `url`.

Application-specific

As in the course of the project at hand not only self-developed applications but also standard applications with known vulnerabilities were tested, the following listing gives an overview of vulnerability IDs covering standard applications:

- Apache web server: 101, 102
- Collabtive: 302
- Openfire: 305, 906
- PhpBB2: 303
- PhpMyAdmin: 402
- Webnews: 501, 703
- Wordpress: 106, 108, 903
4.2 General description of evaluated web application firewalls

The following section covers an analysis of the web application firewall products evaluated during the project. All products are described by a general overview as well as from a technical point of view. The technical part includes the basic configuration as well as the handling of the test cases described above. The protection of the vulnerable test applications has been analysed in two steps: At first all tests have been carried out by a perl script that automates the exploitation of the known weaknesses and analyses the results. In a second step all test cases have been conducted by hand to be able to analyse the reaction of the web application firewalls and to detect possible false positives or false negatives of the script.

4.2.1 phion airlock

phion airlock is a web application firewall that works independently from the web or application server. Therefore phion airlock is compatible with every web server.

Introduction and product description

The company Visonys AG, which was initially developing airlock, was bought by phion in 2008. The phion airlock web application firewall provides an application security gateway, which protects web applications against attacks. The product is completely independent from application or web servers, so security enforcement tasks are outsourced. phion airlock is a software appliance, which makes it possible to also run it in a virtualized environment. As the analyzing tasks are very load intensive, dedicated hardware is recommended. The system is based on Sun Solaris 10 Update 3 and is operated as a reverse proxy.

phion airlock offers the following key security features within the security gateway appliance:

- Intrusion prevention
- Request filtering
- Application monitoring
- User authentication enforcement and session handling
- Cross-platform single sign on
- User tracking
- Application level load balancing
- High availability and fail over
- Integration to access and identity management systems, like LDAP, Radius or Microsoft Active Directory

The security-relevant tasks are organized in different modules. Each module provides further enhancement and configuration capabilities of airlocks security and management features. The availability of a specific module depends on the purchased license. The following list describes available modules in greater detail:
- Web application firewall (WAF) module: Protocol validation and rebuilding, character encoding and unicode verification, form protection, response rewriting, response content filter, cookie protection and response URL encryption

- ICAP Module: offers external content filtering through anti-malware services, intrusion prevention systems or data leakage prevention systems

- SOAP/XML Validator Module: provides systematic monitoring, filtering and validation of SOAP/XML web service data traffic

- Graphical reporting module

- Authentication enforcement module: user requests are only forwarded to the application server, when the user is authenticated and authorized on the web application firewall

- SSL VPN service module: allows secure remote access through a webbrowser

Altough airlock offers a wide range of functionality, mainly the WAF module was tested in the course of this project.

phion airlock is not distributed together with hardware appliances. The customer has to provide the required hardware. This way the customer can also upgrade the hardware if more performance is needed. The product guide of phion airlock gives hardware requirement examples for different ranges of applications. Generally the hardware must be capable to run Solaris 10. Also more than one dedicated server is recommended in order that failover and load balancing features can be used.

**Basic configuration** The installation process of phion airlock is well-documented within the administrator guidelines. After the setup process the administration web interface can be reached on the internal network interface. Although the system is also reachable via secure shell (SSH), all configuration and monitoring tasks are covered by the web interface.

Every application is handled by an individual virtual host, so application-specific settings can be made. Every virtual host is accessible from the external, untrusted network. In this way the internal structure of the application and database servers can be protected. These mappings are also configured within the administration interface. A very basic configuration for testing the standard functionality of the web application firewall module is easy to achieve and done within some minutes. The separation between different hosts also favors different web server technologies with different listening ports. All basic testing efforts have been made using standard configuration in block mode. Specific rewrite or protection rules for web requests have not been configured. Only the standard rule sets for web application attacks were enabled. These standard rules cover wide ranges of web application attack vectors. The rules are built with regular expressions, like in other security appliances. The following listing for example shows the predefined rule for preventing SQL injections:

```
1 (/\*\*|[^&\#]-|-|\.;*(exec(sqlite)|insert|update|select|delete|drop))['[:cntrl
    ...:]][[:space:]][or|and|having|select]('[[:cntrl][:space:]])
```

These predefined rules are also described in the administration interface. The descriptions are useful because they represent a readable version of the matching request contents. According to the SQL injection regular expression these scenarios would be:
Detects SQL injection.
- Any style of comment: /*, -- or #
- ; followed by keywords exec, insert, update, select, delete or drop
- separating char followed by (or, and, having or select) followed by a
  ...separating char

Using the included logviewer all types of logs can be analyzed. Filters allow to differ
between system-wide logs and application logs. Therefore it is easy to view only the logs of
occurred incidents. Unfortunately, no logs concerning the HTTP protocol rewriting in the
early stages of traffic normalization are made.

All tests during this project were conducted using phion airlock version 4.1-10.41 installed
in VMWare Workstation 6.5.0-118166.

Handling of test cases  The following list covers the results of all test cases where it was
tried to exploit vulnerabilities of web applications using the web application firewall as a
proxy. The numbers and descriptions of the test cases can be found in chapter 4.1.2.
109: User enumeration
   Useraccount enumeration GET: success...
   Useraccount enumeration POST: success...

110: Private Directories
   Disclosure of /administrator: success...
   Disclosure of /admin: success...
   Disclosure of /privat: success...
   Disclosure of /private: success...
   Disclosure of /backup: success...

>>> 200: BRUTE FORCE

201: Login Brute Force
   Brute Force GET success...
   Brute Force POST success...

>>> 300: LOGICAL ERRORS

301: imageapp - delete arbitrary files
   Delete of turtle.jpg success...

302: Collaborative Authentication Bypass
   Authentication Bypass: success...

303: phpBB2 Session Handling Authentication Bypass
   phpBB2 Authentication Bypass: fail...

304: Recursive Output
   Recursive Output GET success...
   Recursive Output POST success...

305: OpenFire auth bypass
   openfire auth bypass: fail...

>>> 400: VIOLATION OF THE MINIMAL PRINCIPLE

401: HTTP TRACE
   TRACE: fail...
   TRACE Filter: active...

402: phpMyAdmin
   phpMyAdmin: success...

403: imageapp - delete www-data owned arbitrary files
   Delete of /tmp/file: fail...

404: Upload of PHP files
   Upload of PHP files and command execution: success...

>>> 500: PATH TRAVERSAL / REMOTE CONTENT INCLUSION
501: Remote File Inclusion
RFI/GET: success...
RFI/POST: success...

502: Remote File Inclusion with Nullbyte
webnews rfi with nullbyte %00: fail...
webnews rfi with nullbyte: ?: success...

503: Path Traversal
path traversal get: fail...
path traversal post: fail...

>>> 600: COMMAND INJECTION

601: Command Execution
ls /var/www/admin: (GET) success...
ls /var/www/admin: (POST) success...

602: CGI-BIN Buffer Overflow
Buffer Overflow: success...

>>> 700: INTERPRETER INJECTION

701: PHP eval code injection
PHP eval code injection GET: success...
PHP eval code injection POST: success...

702: MySQL Injection
MySQL Injection 1/GET: fail...
MySQL Injection 1/POST: fail...

703: Wordpress MySQL Injection
Wordpress MySQL Injection: fail...

>>> 800: WEAK STATE MANAGEMENT

801: Authentication via Cookie
Authentication via Cookie: success...

802: Cookie-Manipulation
Cookie-Manipulation: fail...

803: Predefined Session ID
Predefined Session ID: success...

>>> 900: BROWSER-BASED ATTACKS

901: Cross-site request forgery
XSRF: success...

902: Recursive Output
Recursive Output GET fail...
Recursive Output POST fail...
The following table covers only the blocked traffic and the reasons reported by the web application firewall:

<table>
<thead>
<tr>
<th>ID</th>
<th>Result</th>
<th>Matched rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Vulnerability class: Logical errors</td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>Attack failed (*)</td>
<td>Unknown reason</td>
</tr>
<tr>
<td>305</td>
<td>Attack failed (*)</td>
<td>Unknown reason</td>
</tr>
<tr>
<td>400</td>
<td>Vulnerability class: Violation of the minimal principle</td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>Attack failed</td>
<td>Blocked by HTTP headers settings in mappings</td>
</tr>
<tr>
<td>403</td>
<td>Attack failed</td>
<td>(default 07) Value directory traversal rule</td>
</tr>
<tr>
<td>500</td>
<td>Vulnerability class: Path traversal and inclusion of external content</td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>Attack failed</td>
<td>(default 04) Non-printable characters in values rule</td>
</tr>
<tr>
<td>503</td>
<td>Attack failed</td>
<td>(default 07) Value directory traversal rule</td>
</tr>
<tr>
<td>700</td>
<td>Vulnerability class: Interpreter injection</td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>Attack failed</td>
<td>(default 01) SQL injection rule</td>
</tr>
<tr>
<td>703</td>
<td>Attack failed</td>
<td>(default 01) SQL injection rule</td>
</tr>
<tr>
<td>800</td>
<td>Vulnerability class: Weak state management</td>
<td></td>
</tr>
<tr>
<td>802</td>
<td>Attack failed (*)</td>
<td>Unknown reason</td>
</tr>
<tr>
<td>900</td>
<td>Vulnerability class: Browser-based attacks</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Attack Status</td>
<td>Reason</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>902</td>
<td>Attack failed</td>
<td>(default 02) XSS rule for values</td>
</tr>
<tr>
<td>903</td>
<td>Attack failed</td>
<td>(default 02) XSS rule for values</td>
</tr>
<tr>
<td>904</td>
<td>Attack failed (*)</td>
<td>Unknown reason</td>
</tr>
<tr>
<td>905</td>
<td>Attack failed</td>
<td>(default 02) XSS rule for values</td>
</tr>
<tr>
<td>906</td>
<td>Attack failed</td>
<td>(default 02) XSS rule for values</td>
</tr>
</tbody>
</table>

(*) Unknown reason: The log file shows no matching rule as reason for blocking. The reason most likely lies in the interaction of LWP/WWW::Mechanizes cookie handling procedures and airlocks HTTP request handling engine. According to the administration guide, the multi-stage filter 3: Protocol Validation and Rebuilding, which is self-controlled by phion airlock, could be the reason for blocking the request.

Regardless of the specific vulnerability classes, totally 24 of the 65 attacks were blocked by the standard configuration of the web application firewall. This results in a blocking rate of 36.92% as far as these specific test cases are concerned.

**Conclusion** phion airlock is a feature-rich software appliance with various possibilities to enhance web security. One of the biggest advantages of this web application firewall is the user-friendly web interface. An exception of that is the logviewer which is designed in a complicated way. In it some minor problems occurred because not all filtered requests were listed, so this feature can still be improved. Although the goal of phion airlock is to provide a high security software solution, it is interesting to recognise that passwords for the management interface are cut after eight characters. Beside that an interesting fact is, that the administration interface is not covered by the standard rules of the web application firewall. This behavior can be recognised by entering arbitrary (attack) strings within the administration interface that were otherwise filtered on all protected virtual hosts. All tests concerning phion airlock were conducted without major problems.
4.2.2 Artofdefence Hyperguard

Hyperguard is a software-based web application firewall that is intended to be installed directly on the web server to be protected. It acts as a plugin that integrates into the web server.

**Introduction and product description**

Web servers supported by Hyperguard include current versions of Apache, IIS, J2EE-compliant application servers (like Tomcat) or lighttp. From a technical point of view, Hyperguard can be used in connection with any web server because it can also be used as a reverse proxy, as has been done in the project at hand. In that case the reverse proxy runs an instance of an Apache web server where the web application firewall can be integrated. The web server of the reverse proxy is configured to redirect all requests to the real web server using the `Proxy` directive in the configuration.

Artofdefence does not sell hardware appliances in connection with the web application firewall. Appropriate hardware must be provided by the customer (as a dedicated reverse proxy or directly on the existing web server). The software can be downloaded without restriction after registration. Installation packages for a wide range of operating systems and package management systems are provided (Windows-, Linux- and BSD-Systems). Licenses that are needed for a complete installation have to be bought and are issued on a per-IP basis. The IP address of the web server respectively the proxy server has to be entered before the license is issued and cannot be changed afterwards. The license file is then provided as a signed PEM file.

Hyperguard has a modular design that enables different components to be operated on different servers and therefore work in cluster environments. It is for example possible to protect several web servers using multiple instances of Hyperguard on different machines while all configuration and monitoring can be done on a single master host. Hyperguard is divided into three components:

- **Webinterface for administration purposes.** This component enables to monitor and configure all Hyperguard instances.
- **Enforcer plugin.** This component runs on every web server or reverse proxy that has the function of protecting web applications. All HTTP requests and responses are intercepted by the enforcer and forwarded to the decider plugin. Depending on the decision of the decider plugin the enforcer either blocks, forwards or rewrites the message.
- **Decider plugin.** The decider represents the system of rules that is applied to each request processed by the enforcer. The rules can be created or edited using the webinterface.

In course of this project all three components have been installed on a single reverse proxy.

**Basic configuration**

After the installation the administrative functions of the web application firewall can be accessed using the web interface. By default the access to it is restricted to localhost. At first hosts and applications have to be registered in order to configure the protective functions for them. A host consists of an IP address or a host name resolved by a DNS server. An application is defined by one or more web server paths (prefixes) like `myapplication/*`. The prefixes include wildcards and can also be set to the web server root directory, if all published contents of the web server shall be protected by the web application firewall.
The configuration follows an inheritance principle meaning that all security functions defined for a prefix (for example the root directory: /*) do also apply for files and directories below this directory (for example: /myapplication/*). General configuration settings that should apply to all web server contents should therefore be done preferably at the highest level as the settings there will be inherited to deeper directories. All settings of the deeper directories can be customized (individual breakup of the inheritance) if there is the need to do so. Hyperguard logs all requests regardless whether they were allowed or denied. Rule refinements be done directly out of the recorded logs. If for example a legitimate request has been wrongfully blocked, the log viewer offers an option to directly change the ruleset to let similar requests pass in future times.

The configuration of security functions is done via handlers. A handler is a control mechanism that specifies whether certain HTTP requests or responses will be passed or blocked or alternatively performs changes on the messages passing through. A handler is assigned to a prefix (and is inherited as described above). Hyperguard offers 39 handlers that realize certain security functions. Some of them can be configured in great detail whereas others can only be activated. To demonstrate the handler concept the following list describes some handlers and their functions:

- **SessionHandler**: If this handler is activated, Hyperguard applies its own session management between the proxy and the client. All session cookies sent by the server are removed and replaced by random session cookies of the web application firewall. When the client sends subsequent requests, the process is carried out vice versa. An attacker therefore is not able to manipulate session information or values stored in cookies. The session management of this handler also enables to configure session timeouts, connection rate limits, etc.

- **ValidHTTPMethodHandler**: This handler checks whether the HTTP verb in the request header is of an allowed type. By default, only GET, HEAD and POST are allowed.

- **InvalidArgsHandler**: This handler enables to deny requests that contain certain GET or POST parameters that match attack signatures. The parameters (assumed that they are all key-value-paired) are checked using regular expressions. The following regular expression tries for example to detect SQL injections:

\^.*=.*(\'|%27)?.*(;|%3b).*(--|--|%2d%2d)

- **InvalidBodyTextHandler**: This handler is used to filter the contents that are transported from the web server to the client. This enables to filter messages that should not be shown to users. The handler is also configured using regular expressions, for example:

\[Microsoft\]\[ODBC SQL Server Driver\]\[SQL Server\] Syntax error

All handlers offer a default configuration that is intended to deny common attacks against web applications. To achieve a default configuration with the most common security settings a set of handlers and their configuration does not need to be selected manually but can be set up using wizards. Wizards create the needed handlers and their configuration to achieve certain desired security goals. Commonly used wizards are Secure session wizard, SQL injection wizard, or the Baseline protection wizard. The configuration of the created handlers can afterwards be changed manually.
The evaluation of the test cases has been done using only the default ruleset generated by the wizards. By tailoring the configuration more attacks can be prevented but therefore the existing vulnerabilities must be known and understood which is not the case in many real-world scenarios.

**Handling of test cases**  The following list covers the results of all test cases where it was tried to exploit vulnerabilities of web applications using the web application firewall as a proxy. The numbers and descriptions of the test cases can be found in chapter 4.1.2.

```
Testing: aod
Date: Sat Jan 10 22:12:17 2009

  HTML Normalization: active...

>>> 100: INFORMATION DISCLOSURE
   101: Directory Listing
       Listing accessible: success...
   102: Server Banner
       Server Banner visible: success...
   103: HTML Comments
       Comment still present: success...
       HTML page accessible: success...
   104: Disclosure of error messages
       Disclosure of connection error: success...
       Disclosure of username: success...
       Disclosure of syntax error: success...
   105: phpinfo();
       Disclosure of phpinfo(): success...
   106: MySQL
       Disclosure of MySQL database layout: success...
   107: Hidden or temporary files
       Access backup.php: success...
       Access backup.php.bak: success...
       Access backup.php.conf: success...
       Access backup.php.tmp: success...
   108: User enumeration wordpress
       Wrong username: success...
       Wrong password: success...
   109: User enumeration
       Useraccount enumeration GET: success...
       Useraccount enumeration POST: success...
   110: Private Directories
       Disclosure of /administrator: success...
       Disclosure of /admin: success...
       Disclosure of /privat: success...
```
Disclosure of /private: success...
Disclosure of /backup: success...

>>> 200: BRUTE FORCE
201: Login Brute Force
   Brute Force GET success...
   Brute Force POST success...

>>> 300: LOGICAL ERRORS
301: imageapp - delete arbitrary files
   Delete of turtle.jpg success...
302: Collaborative Authentication Bypass
   Authentication Bypass: fail...
303: phpBB2 Session Handling Authentication Bypass
   phpBB2 Authentication Bypass: fail...
304: Recursive Output
   Recursive Output GET success...
   Recursive Output POST success...
305: OpenFire auth bypass
   openfire auth bypass: fail...

>>> 400: VIOLATION OF THE MINIMAL PRINCIPLE
401: HTTP TRACE
   TRACE: fail...
   TRACE Filter: active...
402: phpMyAdmin
   phpMyAdmin: success...
403: imageapp - delete www-data owned arbitrary files
   Delete of /tmp/file: fail...
404: Upload of PHP files
   Upload of PHP files and command execution: success...

>>> 500: PATH TRAVERSAL / REMOTE CONTENT INCLUSION
501: Remote File Inclusion
   RFI/GET: fail...
   RFI/POST: fail...
502: Remote File Inclusion with Nullbyte
   webnews rfi with nullbyte %00: fail...
   webnews rfi with nullbyte: ?: fail...
503: Path Traversal
path traversal get: fail...
path traversal post: fail...

>>> 600: COMMAND INJECTION

601: Command Execution
   ls /var/www/admin: (GET) success...
   ls /var/www/admin: (POST) success...

602: CGI-BIN Buffer Overflow
   Buffer Overflow: success...

>>> 700: INTERPRETER INJECTION

701: PHP eval code injection
   PHP eval code injection GET: success...
   PHP eval code injection POST: success...

702: MySQL Injection
   MySQL Injection 1/GET: fail...
   MySQL Injection 1/POST: fail...

703: Wordpress MySQL Injection
   Wordpress MySQL Injection: fail...

>>> 800: WEAK STATE MANAGEMENT

801: Authentication via Cookie
   Authentication via Cookie: fail...

802: Cookie-Manipulation
   Cookie-Manipulation: fail...

803: Predefined Session ID
   Predefined Session ID: success...

>>> 900: BROWSER-BASED ATTACKS

901: Cross-site request forgery
   XSRF: success...

902: Recursive Output
   Recursive Output GET success...
   Recursive Output POST success...

903: wordpress XSS
   wordpress XSS urlenc: fail...
   wordpress XSS junk: fail...
   wordpress XSS charcode: fail...

904: Cookie-Manipulation - XSS
   Cookie-Manipulation - XSS: fail...
162 | 905: Command Execution XSS
163 | XSS: (GET)
164 | Command Execution urlenc: fail...
165 | Command Execution junk: success...
166 | Command Execution charcode: success...
167 | XSS: (POST) fail...
168 | 906: OpenFire XSS
169 | Openfire XSS urlenc: fail...
170 | Openfire XSS junk: success...
171 | Openfire XSS charcode: success...
172 | Success: 42
173 | Fail: 23

The following table covers only the blocked traffic and the reasons reported by the web application firewall:

<table>
<thead>
<tr>
<th>ID</th>
<th>Result</th>
<th>Matched rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Vulnerability class: Logical errors</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>Attack failed (*)</td>
<td>BaselineProtectionHandler</td>
</tr>
<tr>
<td>303</td>
<td>Attack failed (*)</td>
<td>CookieJarHandler</td>
</tr>
<tr>
<td>305</td>
<td>Attack failed (*)</td>
<td>ValidRequestHandler</td>
</tr>
<tr>
<td>400</td>
<td>Vulnerability class: Violation of the minimal principle</td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>Attack failed</td>
<td>ValidHTTPMethodHandler</td>
</tr>
<tr>
<td>403</td>
<td>Attack failed</td>
<td>BaselineProtectionHandler</td>
</tr>
<tr>
<td>500</td>
<td>Vulnerability class: Path traversal and inclusion of external content</td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>Attack failed</td>
<td>BaselineProtectionHandler</td>
</tr>
<tr>
<td>502</td>
<td>Attack failed</td>
<td>BaselineProtectionHandler</td>
</tr>
<tr>
<td>503</td>
<td>Attack failed</td>
<td>BaselineProtectionHandler</td>
</tr>
<tr>
<td>700</td>
<td>Vulnerability class: Interpreter injection</td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>Attack failed</td>
<td>InvalidArgsHandler</td>
</tr>
<tr>
<td>703</td>
<td>Attack failed</td>
<td>InvalidArgsHandler</td>
</tr>
<tr>
<td>800</td>
<td>Vulnerability class: Weak state management</td>
<td></td>
</tr>
<tr>
<td>801</td>
<td>Attack failed</td>
<td>SessionHandler</td>
</tr>
<tr>
<td>802</td>
<td>Attack failed</td>
<td>SessionHandler</td>
</tr>
<tr>
<td>900</td>
<td>Vulnerability class: Browser-based attacks</td>
<td></td>
</tr>
<tr>
<td>903</td>
<td>Attack failed</td>
<td>InvalidArgsHandler</td>
</tr>
</tbody>
</table>
The exploitation of this attack has been prohibited by the Apache web server configured to operate as an reverse proxy. The web application firewall has nevertheless according functions to also intercept the exploitation attempt.

Regardless of the specific vulnerability classes, totally 23 of the 65 attacks were blocked by the web application firewall. This results in a blocking rate of 35.38% as far as these specific test cases are concerned.

**Conclusion**  The Hyperguard web application firewall has a modular design and offers a rich featureset that can be used to protect web application firewalls. Although the cluster features have not been tested in this project, the possibility to scale the web application firewall functions and operate different components of the firewall on different servers makes is possible to use it in enterprise environments. The product itself offers useful features like versioning of the configuration, monitoring of HTTP traffic to generate positive rulesets or import existing rules from ModSecurity.

In order to configure the web application firewall it takes a not insignificant amount of time as the administrator needs to understand the concept of hosts, applications, prefixes, handlers, configurations and the respective inheritance functions involved. While useful documentation is provided it although takes time to understand the concepts involved. The wizards are in this case very useful as they configure the handlers for the most common security settings.

While Hyperguard provides functions that can indeed prevent the exploitation of web-related bugs, there are also several functions that do not provide real value from a security point of view. For example there are handlers that check the user agent or the referrer of the client. This should reject spiders or automated attack scripts that send a user agent value of their respective scripting language like Python or Perl and not a user agent of a known browser. As the user agent can also easily be manipulated for scripts, this function provides little or no real security value. The only argument that can be used to enable such functions is that they may provide an additional layer to slow down the finding of a flaw or the development of an exploit.

As far as the rulesets provided by the web application firewall are concerned, the according regular expressions are not always effective. In several cases only standard and commonly used attack strings are blocked whereas slightly modified or similar requests are allowed. Chapter 4.3 discusses those issues in greater detail.

During the setup and configuration of the web application firewall several problems occurred that showed that the deployment of a web application firewall does not only solve security problems but can also be the cause of problems. As the project team used a new version of Hyperguard (v3.0) that was released shortly before the evaluation started, the installation package was corrupted. Errors did not occur immediately but instead during the operation of the web application firewall when the ruleset was set active or the system was restarted. A working version of the software was provided after establishing contact with the vendor.

Another problem occurred in connection with file uploads to the web server where the web application firewall at the reverse proxy crashed. The problem was caused by default check routines for SQL injections and was also solved by the vendor after contacting them.
Even if all problems were solved, it was shown that web application firewall technology can also cause problems as far as availability is concerned as there is an additional layer between the client and the web server.
4.2.3 F5 Networks BIG-IP

BIP-IP from F5 Networks is a network appliance that was initially designed for load-balancing. Nowadays many additional software modules are available, such as the Application Security Manager (ASM), which represents a web application firewall implementation from F5.

Introduction and product description  BIG-IP only comes in combination with hardware from F5. In this project the distributor Triple AcceSSS IT generously provided a BIP-IP 3600 appliance in version 9.4.6 Build 401.0 Final for evaluation.

BIG-IP offers a wide variety of features, basically focusing on traffic management. The following product modules are available:

- Local Traffic Manager (LTM)
- Global Traffic Manager (GTM)
- Link Controller (LC)
- Application Security Manager (ASM)
- Secure Access Manager (SAM) standalone version
- WebAccelerator (WA)

The core functionality of these features is implemented in the Traffic Management Operating System (TMOS). As traffic management in general was not in the scope of this project, the analysis efforts focused on the ASM and its features to protect vulnerable web applications.

Basic configuration  In order to activate the protective functions, first of all a basic network configuration of BIG-IP has to be conducted:

- Configure management IP and hostname.
- Create local VLANs.
- Configure IP addresses in the VLANs.
- Assign the VLANs to network interfaces.
- Configure network routes, if needed.

The first step in the further configuration is to define a local traffic pool. The local traffic pool contains the resources that host the actual web application content that is protected with the security policy. After that a virtual server has to be defined. The virtual server processes the incoming traffic, which includes applying the application security class to the transmitted HTTP requests and responses. In our test scenario the virtual server was assigned an official IP address and is linked to the previously configured local traffic pool. This step finishes the basic configuration as so far no web application security settings have been set up.

Further configuration has to be done in the Application Security menu of the web interface. BIG-IP supports the protection of different web applications (virtual hosts) where different rulesets and custom settings per web application can be applied. As a first step an application security class has to be created. This is the logical link between the local traffic components...
and the application security components. BIG-IP uses the application security class to specify
to which incoming HTTP traffic the system applies application security before the virtual
server forwards the traffic to the web application. When configuring an application security
class, the system automatically creates a default web application and a corresponding security
policy on the Application Security Manager. For further information regarding the initial
setup please read [7].

The further configuration has not been accomplished using the automatic configuration
mode. The automatic configuration mode uses the Policy Builder to create the security policy,
which results in the generation of a whitelist based on processed HTTP requests during a
learning phase. As it was not the goal to test the learning mode of the ASM at this time of
the project, a manual setup was done. Creating the configuration in this way results in the
fact that the BIG-IP ASM web application firewall uses a blacklist approach with forbidden
parameter contents suitable for any web application and not adapted for the projects test
cases. This approach also makes the results of the testing affords more easily comparable to
the results of other vendors as their products operate using a blacklist approach per default.
The following configuration steps were necessary to activate the blacklist filtering approach:

- Wildcard symbols for objects, object types and parameters were added. Values for these
  three attributes are normally detected during the learning phase of the web application
  firewall, where the product extracts which pages (objects) exist and with which param-
  eters they are accessed normally. As no objects and their invocation details have been
  learned, they have been replaced by wildcards. Otherwise the ASM would block all
  traffic as it does not know anything about the web applications structure and allowed
  parameters.

- Attack signatures are known attack patterns that the web application firewall uses
to detect malicious traffic. They are applied to all contents of HTTP requests and
responses. The following default signatures were assigned to our policy:
  - All Signatures
  - Generic detection Signatures
  - Systems: PHP, Apache/NCSA, MySQL

This configuration includes nearly all available signatures (excluding those for Microsoft
IIS as the vulnerable web server uses Apache) and results in 1771 enabled signatures. Fur-
thermore the ASM brings out-of-the box protection for common applications such as Outlook
Web Access, Lotus Domino Mail Server, Oracle E-Business Financials, and Microsoft Office
SharePoint along. These policies were not tested within this project.

According to the documentation (see [7]), the attack signatures are able to detect the
following attack types: buffer overflow, directory indexing, authentication/authorization at-
tacks, information leakage, predictable resource location, command execution, vulnerability
scan, Denial of Service, Trojan/Backdoor/Spyware, abuse of functionality, XSS, server-side
code injection, SQL injection, detection evasion, path traversal and LDAP injection. In ad-
dition it is also possible to create user-defined attack signatures if an organisation is aware
of a security flaw that is not covered by the default ruleset and can or will not be patched
directly in the source code.
Handling of test cases  The following list covers the results of all test cases where it was tried to exploit vulnerabilities of web applications using the web application firewall as a proxy. The numbers and descriptions of the test cases can be found in chapter 4.1.2.

1. Testing: bigip
3. HTML Normalization: inactive...

>>> 100: INFORMATION DISCLOSURE

101: Directory Listing
Listing accessible: fail...

102: Server Banner
Server Banner visible: fail...

103: HTML Comments
Comment still present: success...
HTML page accessible: success...

104: Disclosure of error messages
Disclosure of connection error: fail...
Disclosure of username: fail...
Disclosure of syntax error: fail...

105: phpinfo();
Disclosure of phpinfo(): fail...

106: MySQL
Disclosure of MySQL database layout: success...

107: Hidden or temporary files
Access backup.php: fail...
Access backup.php.bak: fail...
Access backup.php.conf: fail...
Access backup.php.tmp: fail...

108: User enumeration wordpress
Wrong username: success...
Wrong password: success...

109: User enumeration
Useraccount enumeration GET: success...
Useraccount enumeration POST: success...

110: Private Directories
Disclosure of /administrator: fail...
Disclosure of /admin: success...
Disclosure of /privat: success...
Disclosure of /private: success...
Disclosure of /backup: fail...

>>> 200: BRUTE FORCE
201: Login Brute Force
   Brute Force GET success...
   Brute Force POST success...

>>> 300: LOGICAL ERRORS

301: imageapp - delete arbitrary files
   Delete of turtle.jpg success...

302: Collaborative Authentication Bypass
   Authentication Bypass: success...

303: phpBB2 Session Handling Authentication Bypass
   phpBB2 Authentication Bypass: fail...

304: Recursive Output
   Recursive Output GET success...
   Recursive Output POST success...

305: OpenFire auth bypass
   openfire auth bypass: fail...

>>> 400: VIOLATION OF THE MINIMAL PRINCIPLE

401: HTTP TRACE
   TRACE: fail...
   TRACE Filter: active...

402: phpMyAdmin
   phpMyAdmin: fail...

403: imageapp - delete www-data owned arbitrary files
   Delete of /tmp/file: fail...

404: Upload of PHP files
   Upload of PHP files and command execution: success...

>>> 500: PATH TRAVERSAL / REMOTE CONTENT INCLUSION

501: Remote File Inclusion
   RFI/GET: fail...
   RFI/POST: fail...

502: Remote File Inclusion with Nullbyte
   webnews rfi with nullbyte %00: fail...
   webnews rfi with nullbyte: ?: fail...

503: Path Traversal
   path traversal get: fail...
   path traversal post: fail...

>>> 600: COMMAND INJECTION
601: Command Execution
ls /var/www/admin: (GET) fail...
ls /var/www/admin: (POST) fail...

602: CGI-BIN Buffer Overflow
Buffer Overflow: success...

>>> 700: INTERPRETER INJECTION

701: PHP eval code injection
PHP eval code injection GET: fail...
PHP eval code injection POST: fail...

702: MySQL Injection
MySQL Injection 1/GET: fail...
MySQL Injection 1/POST: fail...

703: Wordpress MySQL Injection
Wordpress MySQL Injection: fail...

>>> 800: WEAK STATE MANAGEMENT

801: Authentication via Cookie
Authentication via Cookie: success...

802: Cookie-Manipulation
Cookie-Manipulation: fail...

803: Predefined Session ID
Predefined Session ID: success...

>>> 900: BROWSER-BASED ATTACKS

901: Cross-site request forgery
XSRF: success...

902: Recursive Output
Recursive Output GET fail...
Recursive Output POST fail...

903: wordpress XSS
wordpress XSS urlenc: fail...
wordpress XSS junk: fail...
wordpress XSS charcode: fail...

904: Cookie-Manipulation - XSS
Cookie-Manipulation - XSS: fail...

905: Command Execution XSS
XSS: (GET)
Command Execution urlenc: fail...
Command Execution junk: fail...
Command Execution charcode: fail...
XSS: (POST) fail...
The following table covers only the blocked traffic and the reasons reported by the web application firewall:

<table>
<thead>
<tr>
<th>ID</th>
<th>Result</th>
<th>Matched rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Vulnerability class: Information Disclosure</td>
<td>index of / response, Directory Listing (2), Directory Listing</td>
</tr>
<tr>
<td>101</td>
<td>Attack failed</td>
<td>index of / response, Directory Listing (2), Directory Listing</td>
</tr>
<tr>
<td>102</td>
<td>Attack failed</td>
<td>PHP Information Leakage (5), SQL-INJ mysql.user (Headers)</td>
</tr>
<tr>
<td>104</td>
<td>Attack failed</td>
<td>/phpinfo.php access</td>
</tr>
<tr>
<td>105</td>
<td>Attack failed</td>
<td>/backup access</td>
</tr>
<tr>
<td>110</td>
<td>Attack failed</td>
<td>/backup access, Web-Server Administrator dir access</td>
</tr>
<tr>
<td>300</td>
<td>Vulnerability class: Logical errors</td>
<td>Cookie encryption</td>
</tr>
<tr>
<td>303</td>
<td>Attack failed</td>
<td>Directory traversals</td>
</tr>
<tr>
<td>400</td>
<td>Vulnerability class: Violation of the minimal principle</td>
<td>Blocked Request Type</td>
</tr>
<tr>
<td>401</td>
<td>Attack failed</td>
<td>/phpmyadmin/ dir access (/phpmyadmin/)</td>
</tr>
<tr>
<td>403</td>
<td>Attack failed</td>
<td>Directory Traversal attempt (parameter)</td>
</tr>
<tr>
<td>500</td>
<td>Vulnerability class: Path traversal and inclusion of external content</td>
<td>Illegal meta character in parameter value</td>
</tr>
<tr>
<td>501</td>
<td>Attack failed</td>
<td>Generic Remote File/Path Include Attempt 4 (dir param, http/https)</td>
</tr>
<tr>
<td>503</td>
<td>Attack failed</td>
<td>Directory Traversal attempt (parameter)</td>
</tr>
<tr>
<td>600</td>
<td>Vulnerability class: Command Execution</td>
<td>ls execution attempt</td>
</tr>
<tr>
<td>700</td>
<td>Vulnerability class: Interpreter injection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attack failed</td>
<td>Vulnerability class: Weak state management</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>701</td>
<td>PHP injection attempt (system)</td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>Attack failed</td>
<td>SQL-INJ expressions like or 1=1 (3), Comments (3)</td>
</tr>
<tr>
<td>703</td>
<td>Attack failed</td>
<td>SQL-INJ UNION SELECT (Parameter), SQL-INJ SELECT CONCAT(), SQL-INJ null,null,null, SQL-INJ CHAR(), Comments (3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Attack failed</th>
<th>Vulnerability class: Browser-based attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>Modified domain cookie(s)</td>
<td></td>
</tr>
<tr>
<td>802</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>900</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>902</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>903</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>904</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>905</td>
<td>Attack failed</td>
<td>echo execution attempt, XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
<tr>
<td>906</td>
<td>Attack failed</td>
<td>XSS script tag (Parameter), alert() (Parameter)</td>
</tr>
</tbody>
</table>

Regardless of the specific vulnerability classes, totally 44 of the 65 attacks were blocked by the web application firewall. This results in a blocking rate of 67.69% as far as these specific test cases are concerned.

**Conclusion**  The Application Security Module from BIG-IP is a sophisticated and feature-rich web application firewall. As can be seen, the default blocking rate as far as the specific test cases are concerned is very high (highest percentage of all tested web application firewalls in this project). The manual configuration is done within a short time and the pre-defined attack signatures cover the test cases at a very high level. With 1771 default attack signatures enabled by default the web application firewall offers far the most default signatures of all products. Compared to other web application firewalls these rules are much more detailed. Due to their separation in different groups, an easy selection of active rules can be made. There is also the possibility to assign different modes (blocking or reporting) to these enabled rules.

The signatures not only cover the detection of generic attacks (such as the inclusion of script code used for cross-site scripting) but also signatures for products or systems that are commonly used in the field of web applications and their management. The default ruleset for example blocks access to *phpmyadmin* which is used for database management. While the use of *phpmyadmin* per se is not a security problem it is an application that should only be used by administrators and public access to it most likely is not wanted. If the web site operators are not aware of the fact that the application is publicly accessible via the Internet, it may not be integrated into the patch management process and may therefore be vulnerable to known or not yet publicly released security flaws. Therefore ASM blocks access by default. This approach and the high number of default signatures also lead to the fact that comprehensive
testing has to be conducted after integrating the web application firewall because there is a high possibility that legitimate requests are blocked.

It is worth mentioning that the ASM does not work with first matching. It completes the testing of all configured attack signatures even if a first match is already found. That means that an administrator not only sees one reason why a request has been blocked but also sees all other signatures which would also have blocked the request. Generally the reporting mechanisms meet even high demands and can be qualified as very well-engineered. This is a valuable feature when it comes to debugging and troubleshooting. For example, when a block occurs, an alphanumeric error string is given to the user. With this string the administrator can identify the problem promptly and support the user within a short time. These features allow an easy adjustment of rules and threshold values, so false positives can be corrected easily.

Additionally BIG-IP ASM is the only web application firewall product that operates using a learning mode by default. While other vendors also offer learning features, they are not used by default. The learning mode analyses real time traffic and sets limits due to the user behavior during the learning phase. The resulting ruleset can later also be adapted by the ASM administrator.
4.2.4 Breach Security ModSecurity

ModSecurity is an open source web application firewall that runs as an Apache module. It can work either embedded or as a reverse proxy and provides protection from a range of attacks against web applications and allows HTTP traffic monitoring, logging and real-time analysis.

Introduction and product description ModSecurity was originally developed by Ivan Ristić. He designed it as a means to obtain a proper audit log, but it grew to include other security features. In 2006 ModSecurity was purchased by Breach Security, who decided to continue the open source development. The company now also offers a set of appliances based on ModSecurity, commercial support and ModSecurity training. In short, ModSecurity does the following:

- Intercepts HTTP requests before they are fully processed by the web server
- Intercepts the request body (e.g., the POST payload)
- Intercepts, stores, and optionally validates uploaded files
- Performs anti-evasion actions automatically
- Performs request analysis by processing a set of rules defined in the configuration
- Intercepts HTTP responses before they are sent back to the client
- Performs response analysis by processing a set of rules defined in the configuration
- Takes one of the predefined actions or executes an external script when a request or a response fails analysis (a process called detection)
- Depending on the configuration, a failed request may be prevented from being processed and a failed response may be prevented from being seen by the client (a process called prevention)
- Performs audit logging

As ModSecurity itself is just an Apache module it is not possible to directly protect other web servers such as Microsoft IIS. Though if ModSecurity is configured within an reverse proxy built on top of Apache it is feasible to protect any web server behind it. This scenario was used in the project at hand. In effect ModSecurity is nothing else than a filter for HTTP traffic with a set of regular expressions and a simple logging functionality. Although various projects enhance the handling and functionality:

- Breach Security offers a webbased console to collect logs and alerts from ModSecurity sensors.
- Christian Bockermann wrote a number of tools written in Java in the field of ModSecurity. The following and other several tools and libraries can be found at http://www.jwall.org:

---

4Ivan Ristić is a web security specialist and author of the book *Apache Security* (see: [10]).
5Reverse proxy with Apache 2.3.2 on Debian GNU/Linux 4.0
6Free of charge up to 3 sensors.
AuditViewer provides a GUI to analyze logfiles from ModSecurity. It is possible to modify recorded events and re-inject them.

WebApplicationProfiler is an attempt to extract a web applications profile from recorded audit-data and build a ruleset in terms of a positive security (whitelist) model.

REMO is a project to build a graphical rule editor with a positive/whitelist approach. REMO is currently not developed, the last version was released in June 2007. It can be found at http://remo.netnea.com.

**Basic configuration**  
Installation of ModSecurity can be done without much time and effort. Most current Linux distributions contain ModSecurity in their repositories. An installer for Microsoft Windows is also available. Otherwise the compilation is straight-forward and was done in the project to test the latest release. ModSecurity comes with a minimal configuration without any sample rules. Breach Security offers a core ruleset on their website which focuses on the most common web security vulnerabilities. This ruleset is widely used and was also applied in this project. Community-developed rulesets can be found at http://www.gotroot.com. They are more application- and malware-specific and are interesting for further work beyond this project.

A sample ModSecurity rule looks like follows:

```
SecRule REQUEST_METHOD "!^((?:(?:POS|GE)T| OPTIONS\|HEAD ))$" \\
"phase:2,t:none,log,auditlog,status:501,msg:'Method is not allowed by policy ...
...', severity:'2',id:'960032',tag:'POLICY/METHOD_NOT_ALLOWED'
```

This rule checks the HTTP method in the HTTP header. It only allows GET, POST, OPTIONS and HEAD. The TRACE method, for example, is disallowed as it is excluded in the regular expression. This reflects on line 80 of the output of the test cases script. The second line of the rule tells ModSecurity when to check (phase 2 means after URI translation and header parsing) and what to do if the rule matches. Accordingly, when using the HTTP TRACE method, ModSecurity will return the following lines to the client:

```
HTTP/1.1 501 Method Not Implemented
Allow: TRACE
Content-Length: 327
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

By default ModSecurity logs only to a single file. To get more detailed information and gain clarity it is suggested to enable concurrent logging and adapt the logging configuration to your needs.

**Handling of test cases**  
The following list covers the results of all test cases where it was tried to exploit vulnerabilities of web applications using the web application firewall as a proxy. The numbers and descriptions of the test cases can be found in chapter 4.1.2.

---

7 January 2009: ModSecurity 2.5.7
8 Further information about building rules can be gathered at [4].
9 See [4, Section: Configuration Directives].
Testing: modsec

Date: Sun Jan 11 11:50:16 2009

HTML Normalization: active...

>>> 100: INFORMATION DISCLOSURE

101: Directory Listing
Listing accessible: fail...

102: Server Banner
Server Banner visible: fail...

103: HTML Comments
Comment still present: success...
HTML page accessible: success...

104: Disclosure of error messages
Disclosure of connection error: fail...
Disclosure of username: fail...
Disclosure of syntax error: fail...

105: phpinfo();
Disclosure of phpinfo(): success...

106: MySQL
Disclosure of MySQL database layout: success...

107: Hidden or temporary files
Access backup.php: success...
Access backup.php.bak: fail...
Access backup.php.conf: fail...
Access backup.php.tmp: success...

108: User enumeration wordpress
Wrong username: success...
Wrong password: success...

109: User enumeration
Useraccount enumeration GET: success...
Useraccount enumeration POST: success...

110: Private Directories
Disclosure of /administrator: success...
Disclosure of /admin: success...
Disclosure of /privat: success...
Disclosure of /private: success...
Disclosure of /backup: success...

>>> 200: BRUTE FORCE

201: Login Brute Force
Brute Force GET success...
Brute Force POST success...
>>> 300: LOGICAL ERRORS
301: imageapp - delete arbitrary files
   Delete of turtle.jpg success...
302: Collaborative Authentication Bypass
   Authentication Bypass: fail...
303: phpBB2 Session Handling Authentication Bypass
   phpBB2 Authentication Bypass: fail...
304: Recursive Output
   Recursive Output GET success...
   Recursive Output POST success...
305: OpenFire auth bypass
   openfire auth bypass: fail...

>>> 400: VIOLATION OF THE MINIMAL PRINCIPLE
401: HTTP TRACE
   TRACE: fail...
   TRACE Filter: active...
402: phpMyAdmin
   phpMyAdmin: success...
403: imageapp - delete www-data owned arbitrary files
   Delete of /tmp/file: fail...
404: Upload of PHP files
   Upload of PHP files and command execution: success...

>>> 500: PATH TRAVERSAL / REMOTE CONTENT INCLUSION
501: Remote File Inclusion
   RFI/GET: fail...
   RFI/POST: fail...
502: Remote File Inclusion with Nullbyte
   webnews rfi with nullbyte %00: fail...
   webnews rfi with nullbyte?: fail...
503: Path Traversal
   path traversal get: success...
   path traversal post: success...

>>> 600: COMMAND INJECTION
601: Command Execution
   ls /var/www/admin: (GET) fail...
   ls /var/www/admin: (POST) fail...
602: CGI-BIN Buffer Overflow
Buffer Overflow: success...

>>> 700: INTERPRETER INJECTION

701: PHP eval code injection
PHP eval code injection GET: success...
PHP eval code injection POST: success...

702: MySQL Injection
MySQL Injection 1/GET: fail...
MySQL Injection 1/POST: fail...

703: Wordpress MySQL Injection
Wordpress MySQL Injection: fail...

>>> 800: WEAK STATE MANAGEMENT

801: Authentication via Cookie
Authentication via Cookie: success...

802: Cookie-Manipulation
Cookie-Manipulation: success...

803: Predefined Session ID
Predefined Session ID: success...

>>> 900: BROWSER-BASED ATTACKS

901: Cross-site request forgery
XSRF: success...

902: Recursive Output
Recursive Output GET fail...
Recursive Output POST fail...

903: wordpress XSS
wordpress XSS urlenc: fail...
wordpress XSS junk: fail...
wordpress XSS charcode: fail...

904: Cookie-Manipulation - XSS
Cookie-Manipulation - XSS: fail...

905: Command Execution XSS
XSS: (GET)
Command Execution urlenc: fail...
Command Execution junk: fail...
Command Execution charcode: fail...
XSS: (POST) fail...

906: OpenFire XSS
Openfire XSS urlenc: fail...
Openfire XSS junk: fail...
Openfire XSS charcode: fail...
The following table covers only the blocked traffic and the reasons reported by the web application firewall:

<table>
<thead>
<tr>
<th>ID</th>
<th>Result</th>
<th>Matched rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td><strong>Vulnerability class: Information disclosure</strong></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Attack failed</td>
<td>Leakage Information: Directory Listing</td>
</tr>
<tr>
<td>102</td>
<td>Attack failed</td>
<td>ModSecurity fakes Server Banner</td>
</tr>
<tr>
<td>104</td>
<td>Attack failed</td>
<td>Leakage Information: PHP Information Leakage</td>
</tr>
<tr>
<td>107</td>
<td>Attack failed</td>
<td>Extended Policy: URL file extension is restricted by policy</td>
</tr>
<tr>
<td>300</td>
<td><strong>Vulnerability class: Logical errors</strong></td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>Attack failed (*)</td>
<td>HTML Normalization from Proxy</td>
</tr>
<tr>
<td>303</td>
<td>Attack failed (*)</td>
<td>Normalization form Proxy</td>
</tr>
<tr>
<td>305</td>
<td>Attack failed (**)</td>
<td>Path Traversal Attack</td>
</tr>
<tr>
<td>400</td>
<td><strong>Vulnerability class: Violation of the minimal principle</strong></td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>Attack failed</td>
<td>Request Method is not allowed by policy and XSS Attack (401-2)</td>
</tr>
<tr>
<td>402</td>
<td>Attack failed</td>
<td>Path Traversal Attack</td>
</tr>
<tr>
<td>500</td>
<td><strong>Vulnerability class: Path traversal and inclusion of external content</strong></td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>Attack failed</td>
<td>Remote File Inclusion Attack</td>
</tr>
<tr>
<td>502</td>
<td>Attack failed</td>
<td>Invalid character in request and RFI</td>
</tr>
<tr>
<td>600</td>
<td><strong>Vulnerability class: Command Injection</strong></td>
<td></td>
</tr>
<tr>
<td>601</td>
<td>Attack failed</td>
<td>System Command Injection</td>
</tr>
<tr>
<td>700</td>
<td><strong>Vulnerability class: Interpreter injection</strong></td>
<td></td>
</tr>
<tr>
<td>702</td>
<td>Attack failed</td>
<td>SQL Injection Attack</td>
</tr>
<tr>
<td>703</td>
<td>Attack failed</td>
<td>SQL Injection Attack</td>
</tr>
<tr>
<td>900</td>
<td><strong>Vulnerability class: Browser-based attacks</strong></td>
<td></td>
</tr>
<tr>
<td>902</td>
<td>Attack failed</td>
<td>Cross-site Scripting (XSS) Attack</td>
</tr>
<tr>
<td>903</td>
<td>Attack failed</td>
<td>Cross-site Scripting (XSS) Attack</td>
</tr>
<tr>
<td>904</td>
<td>Attack failed</td>
<td>Cross-site Scripting (XSS) Attack</td>
</tr>
</tbody>
</table>
The exploitation of this attack has been prohibited by the Apache web server configured to operate as an reverse proxy.

The exploitation of this attack has been prohibited by the Apache web server configured to operate as a reverse proxy. The web application firewall has nevertheless according functions to also intercept the exploitation attempt.

Regardless of the specific vulnerability classes, totally 34 of the 65 attacks were blocked by the web application firewall. This results in a blocking rate of 52.31% as far as these specific test cases are concerned.

**Conclusion** ModSecurity is an easy to deploy web application firewall which comes free of charge. If the administrator is used to the configuration of Apache web servers he will be able to set up ModSecurity within little time. The project team did not experience any problems running ModSecurity during the test period. Although it comes without any rules, the core ruleset from Breach Security achieves rather impressive results. For example, the regular expressions for preventing cross-site Scripting Attacks cannot be as easily bypassed like the ones of Hyperguard (see section 4.3).

For providing centralized and advanced logging functionality the ModSecurity Console was evaluated. Unfortunately it was not possible to get the Console to read the logfiles from Apache. Due to the lack of documentation, no solution could be found. It is assumed the latest version of Console has different configuration parameters and settings which could not be determined. However, the tools from [http://jwall.org](http://jwall.org) turned out to be valuable and can be recommended. Especially the re-inject feature of AuditViewer is useful for debugging the ruleset and pentesting activities.

Compared to the other evaluated web application firewalls ModSecurity is pretty basic, however powerful at its scope: HTTP traffic filtering with regular expressions. There are no features as graphical rule editing, URL- or Cookie encryption, form protection or advanced graphical reporting. Other abilities such as high availability and load balancing can be built around ModSecurity with open source software but they do not come directly with the web application firewall. Although if the range of operation is only filtering HTTP traffic, ModSecurity is a valuable option.
4.3 Circumvention of default filtering mechanisms

The following section discusses possibilities to circumvent default filtering mechanisms of the tested web application firewalls. The perl script for an automated evaluation of filtering mechanisms developed during this project (see section 4.2) tests the filtering capabilities by trying to exploit previously known and implemented vulnerabilities. As attacks against web applications can typically be conducted using a variety of different means (character encoding, usage of different keywords or functions, obfuscation using comments, etc), the very same attacks can be conducted by a number of differently assembled requests. As web application firewalls typically operate using a blacklist approach and allow all requests that do not match the blacklists, attacks can to some extent be obfuscated and pass the filtering engines.

All attacks that have been marked as blocked by the automated perl script have been analysed manually to determine the effectiveness of the filtering procedures in connection with that specific test case. As not all test cases can be covered here and possibilities for circumvention are partly the same, the following chapter gives an overview of the found options for circumvention. Please note that the bypass of filtering mechanisms if often demonstrated in connection with a web application firewall product. The fact that the issue is shown at the example of a product does not mean that products of other vendors are not also susceptible to the same circumvention technique shown.

In connection with the test case 601 (command execution) the Hyperguard web application firewall does not allow to print the contents of the /etc directory (e.g. cat /etc/passwd). The restriction is only limited on this directory. On the other side an attacker can enumerate all the server content using the ls command and also read files using the cat-command for files the user www-data has access to and that are not in the /etc directory. Blocking access to /etc surely lowers the impact of an attack as several configuration files cannot be easily read, but does not protect other system resources in other directories that can also be used to gather information or sensitive data.

Another example for incompletely implemented regular expressions for filtering is the easy bypass of the cross-site scripting filter mechanism of Hyperguard. In the following listing only the first line is blocked. All other requests are not blocked by the web application firewall and therefore enable an attacker to include arbitrary script code:

```
1 <script>alert(1)</script>
2 <script+abc>alert(1)</script+abc>
3 <script>alert(1)</script>
4 <SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
```

The following example regarding the BIG-IP web application firewall shows clearly that a blacklist-based approach in some cases cannot effectively protect a web application infrastructure. An attack may be slowed down or less experienced attackers using standard exploit mechanisms may be kept off, but the defense is nevertheless insufficient.

The following demonstration is related to test case 601 (command execution), where an attacker is able to inject arbitrary commands that are executed with the privileges of the web server. The affected script enables users to ping hosts by entering an IP address. The given IP address is then passed to the command line tool ping and the results are echoed back to the user. A normal invocation of the according PHP script looks like follows:
If an attacker tries to append additional commands to the parameter, the web application firewall blocks the request. The following request is for example blocked because the `whoami` command matches one of the built-in blacklist filters:

```
cmd_exec.php?ip=4.2.2.1
```

In order to circumvent the filter it is possible to make use of the fact that the Apache web server by default runs with the privileges of an ordinary user (www-data) which has access to technical resources and capabilities like other users or processes. That means that the web server process also has access to environment variables that can be read and written. An attacker can use this fact to write the command to be executed in parts to environment variables and execute them afterwards. The following listing shows how the command `whoami` is split into two parts, written to environment variables and used for command execution:

```
4.2.2.1; a=who; b=ami; $a$b
```

As the request does not match any blacklist filters, it is passed to the web server, where the command is executed (see figure 2).

The same methodology (using environment variables) can be used to bypass the aforementioned restricted access to the /etc directory of the Hyperguard web application firewall. Whereas the first access attempt in the following listing is denied, the second one leads to success and reveals the contents of the systems password file:

```
4.2.2.1; cat /etc/passwd
4.2.2.1; a/etc; cat /$a/passwd
```

Test case 702 ("mysqlinjection_get") offers a login form which is vulnerable to SQL injection attacks. To bypass the login an attacker needs to inject SQL syntax in order to instruct the database to return a valid user even if the passwords do not match. The Hyperguard web application firewall blocks according requests where injected SQL syntax is recognized. The filter can however be bypassed by entering comment characters that are not interpreted by the database but circumvent the blacklist filter. In the following listing the first request is blocked by the web application firewall, but the second one is forwarded to the web server enabling an attacker to log in as `userA` without knowledge of the according password:
Another problem as far as this test case is concerned occurs in connection with the ModSecurity web application firewall where the default ruleset can also be bypassed. The attack makes use of a syntax issue in connection with the MySQL database. Whereas other databases require explicit comparisons (e.g. `or 1=1`) to construct a true statement, MySQL also accepts the following statements as true:

```
1 or 1
2 or TRUE
3 or version()
4 or sin(1)
5 ...
```

Whereas the first request in the following listing is blocked (ModSecurity detects the SQL Syntax because of the single quote in connection with the equality sign), the other requests are passed on to the web server and are successfully processed by the database. The blacklist filter is bypassed because of the missing comparison.

```
1 userA' or 1=1#
2 userA' or 1#
3 userA '+''/#
```

The blacklist filter of phion airlock works according to a multiple keyword matching approach. If a request contains a single quote or an equality sign, the request is not blocked per se. The request is only dropped if it contains both signs at the same time. The same holds true for requests containing SQL comment signs (`--`, `#`, `/*`).

Beside the possibilities to bypass filtering rules that try to mitigate critical vulnerabilities like cross-site scripting, command execution and SQL injection, there are also certain areas where the filtering rules of the tested web application firewalls seem to operate in a reasonable way. The following areas tended to be hard to circumvent:

- Remote and local file inclusion (possibilities to rewrite requests that still have the same meaning are limited in this area)
- Cookie-related vulnerabilities (current web application firewalls replace all cookie contents by a single and randomly chosen cookie)

Whereas some of the blocked vulnerabilities also could not be exploited by rewriting the original requests of the perl script, it could be shown that the blacklist-approach adopted by web application firewalls lacks of full coverage of all possible attack vectors. Because of the huge amount of different encodings, notations and possible syntaxes it is hard to cover all possible attacks. An additional problem the developers of such blacklists face is that with risen coverage of attack vectors also the number of false positives rises. While a general blocking of all special characters (as these are used for command separation or syntax designation in programming languages) would prevent many vulnerabilities from being exploitable, but this would also render many web applications useless because of the high number of false positives.

The conclusion of the circumvention attempts carried out by the project team can be summarized as follows: With a purely blacklist-based approach (as many web application firewalls work today) there is always a balance between the non-effectiveness of the filter.
mechanisms and the number of false positives (and therefore falsely blocked user requests) the operators of a web application infrastructure have to face. Even if a first attempt to exploit is blocked, a 100% coverage of all encoded attacks cannot be achieved.
4.4 Anomaly detection capabilities

Current web application firewall technology as described in the previous chapters mostly works according to a blacklist approach. This means that only those requests are blocked that match the filters. As also has been outlined in the previous chapters, this approach often does not provide effective protection. Therefore vendors of web application firewall technology start to implement mechanisms that analyse the default traffic patterns of web applications. After enough traffic has been collected, the requested pages and sent parameters are analysed and patterns can be extracted. The detailed approach is outlined in academic technical papers which can be found in [8] and [9]. Of the test equipment used during this evaluation only Artodefence Hyperguard and F5 Networks BIG-IP supported those learning modes (other vendors now provide according functions too, but not within the versions that were provided at the start of the evaluation). As both are commercially distributed systems, no details about the implementation of the learning modes could be obtained.

Both systems analyse logfiles of previously received requests and generate whitelist filters that match the known traffic patterns. The filters concern both the documents that can be accessed on the web server (valid URLs) as well as the parameters that can be send to dynamic content of web applications (contents of GET and POST variables). For testing the default filtering mechanisms of the web application firewalls, several vulnerable web applications have been implemented (see chapter 4). As a considerable amount of traffic has to be generated by hand (by using the web application as users normally would) in order to collect enough data to extract patterns, only a subset of the applications was used.

All testing efforts were restricted to the functionality provided under the folder imageapp. The application offers registered users the possibility to upload and share photos. After the login procedure the user sees all currently available photos either uploaded by him- or herself or by other users of the same application. A user can manage the photobase by uploading new images or by deleting existing ones. Implemented vulnerabilities in connection with the application are the deletion of files owned by other users, path traversal, upload of PHP files as well as cross-site request forgery.

For the subsequent practical evaluation of the learning mode capabilities only Artodefence Hyperguard was used. Because of an unrecoverable software failure (database corruption) of the BIG-IP web application firewall that could not be repaired within the time resources left for the project, the web application firewall could not be used for the according milestone.

The Artodefence Hyperguard web application firewall is not strictly separated in a blacklist mode and a learning mode resulting in a whitelist generation. The learning mode is rather an additional wizard that can be executed and should help to additionally tight up the ruleset. That means that already existing blacklist rules are extended by additional whitelist regular expressions. All parameters must match the specifically generated regular expressions in order to be forwarded to the web server. In connection with Hyperguard the wizard configures the InvalidArgsHandler and therein sets so-called valid_key_value_pattern.

The rule generation follows a process of two steps. In the first step the wizard enforces extensive logging of all requests that are sent to the web application firewall and are forwarded to the web server. In high-load environments this process can decrease the performance of the system. After logging has been enabled the wizard exits and waits until enough traffic has been collected. As the evaluation was limited to the application imageapp the project team was able to generate the needed traffic by hand. The application was used in a normal way (logins with different users, logouts, uploads of photos, deletion of own photos, downloads of
Figure 3: Rules suggestion using the learning mode.

After enough data has been collected, the wizard allows to proceed to the second step. Extensive logging is disabled again and all collected requests are analysed for patterns. All learned paths within the application are listed and according parameters are shown. The wizard is to be meant to suggest regular expressions that match the already seen requests (see figure 3).

As can be seen, the wizard for example suggests that the parameter file should be matched against a regular expression that indicates a parameter length between 0 and 32 consisting of alphabetic characters and/or special characters. This parameter is for example used to indicate which image a user wants to delete (using the file delete.php). If for example a user decides to upload an image with a file name longer than 32 characters (though maybe unlikely), the generated regular expression does not match anymore and the application cannot be used.

After the generated rules have been activated the functionality of the application has been tested manually again. The first problem that occurred was that the usual start page could not be accessed anymore because it was not whitelisted. In order to be able to login the PHP file had to be accessed directly (see the following listing):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Blocked: <a href="http://host/imageapp/">http://host/imageapp/</a></td>
</tr>
<tr>
<td>2</td>
<td>Working: <a href="http://host/imageapp/login.php">http://host/imageapp/login.php</a></td>
</tr>
</tbody>
</table>

The second problem was that users were not able to login. Due to unknown reasons the regular expression for the parameter user stated that valid user names have the same layout as email addresses even though the only existing and used users during the learning phase
were userA, userB and userC. The problem was also reproducible during several executions of the wizard. After again rewriting the generated rules users were able to login. The next problem that occurred was that the rules did not allow to display uploaded image files (because the according and new URLs previously were not learned). The application could only be recovered to be functional again by removing a large part of the whitelist rules previously added by the wizard.

Because the imageapp application is only small and does not offer a lot of functionality and given the high number of valid requests that were blocked, the learning feature of Hyperguard (at the evaluated version) cannot be considered stable enough to be used in connection with a productive environment.

While by manually refining the rules it is possible to get the application working again (at the first glance), the project team has the impression that the ruleset is prone to a high number of false positives during productive operations. This is mainly because of length restrictions (e.g. file name lengths of 32 characters) and of normally valid data not occurred during the learning phase. It can of course be argued that the learning phase must be relatively long in order that all valid traffic occurred at least once. However, that rises the possibility that attacks occur during the learning phase that are subsequently whitelisted. Another problem is that the learning mode must be executed again every time the functionality of the protected web application is changed.
4.5 Test of parsing robustness using fuzzing techniques

As already has been outlined in this document, web application firewalls represent additional systems that are placed between clients and web servers processing HTTP requests and responses. The systems therefore add additional complexity to the overall system and can in case of failure or poor configuration be the cause for disrupted service and unavailability. Additionally, an attacker does not necessarily need to find vulnerabilities in the web server or web applications, but can also make use of flaws in the web application firewall.

It es therefore vital that the parsing routines of HTTP requests are implemented in a robust way and also handle requests that do not comply with the HTTP specification in a defined way. It must not be possible for an attacker to influence the operations of the web application firewall in an undesired way or to cause denial of service scenarios by sending specially crafted HTTP requests.

4.5.1 Testing approach

Testing the robustness of software implementations requires either access to the source code or can be achieved in a blackbox approach. As the project team does not have access to the implementation details of all web application firewalls, the blackbox approach was chosen. That requires the generation of a great number of test cases that violate the protocol specification. To cover a preferably broad spectrum of possible requests, a testing approach using fuzzing was chosen.

Fuzzing seeks to detect flaws in software implementations by automatically creating a series of crafted requests, execute the target software using these generated requests and analyse the results or the program behavior. Fuzzers can generally be divided into data generation and data mutation fuzzers. Whereas data generation fuzzers have an implemented model of which data to generate (they „understand“ the protocol or file format to be fuzzed), data mutation fuzzers are more generic in that they take input data (for example a valid file or network request) and randomly change parts in it.

For the project at hand a data generation fuzzing approach was chosen. The project team hereby used the Sulley Fuzzing Framework\(^\text{10}\) as it already implements recurring needs like transmission of test data and fault monitoring. The framework and the according test case descriptions are written in Python.

To implement test cases, HTTP requests are modeled using a block-based approach as described in [1]. That means that HTTP requests are divided into several blocks that are described individually. The descriptions are subsequently interpreted the fuzzer. For each generated request a certain block of the description is chosen and replaced by a new value. The Sulley Fuzzing Framework can both generate random new values or take the values from a preset list containing long strings, format string specifiers, SQL injection strings, etc. Values can also be duplicated or left out. To illustrate the block-based approach, the following listing shows the description for HTTP HEAD requests as it is one of the shortest descriptions used in the project:

```
#!/usr/bin/python
import os
import sys
```

\(^{10}\)See the following URL for further information: [http://www.fuzzing.org/fuzzing-software](http://www.fuzzing.org/fuzzing-software)
from sulley import *

host = "127.0.0.1"
port = 4444
request_type="HTTP_HEAD"

s_initialize(request_type)
if s_block_start(request_type):
    s_string("HEAD ")
    s_delim("/")
    s_string("directory")
    s_delim("/")
    s_string("anysite.jsp")
    s_delim(" ")
    s_string("HTTP")
    s_delim("/")
    s_string("1")
    s_delim(".")
    s_string("0")
    s_static("\r\n")
    s_string("Host ")
    s_delim(":" )
    s_string("webapphost.com")
    s_static("\r\n")
    s_static("User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko ...
    /20070312 Firefox/1.5.0.11 \r\n")
    s_static("Accept: image/gif, image/jpeg, */* \r\n")
    s_static("Connection: close \r\n")
    s_static("\r\n")
    s_block_end()
    target = sessions.target(host, port)
sess = sessions.session(session_filename=os.path.basename(sys.argv[0])+",
.../session", timeout=0.5, skip=0)
sess.add_target(target)
sess.connect(s_get(request_type))
sess.fuzz()

Please note that other descriptions used in this project do not define header values as static
and also contain multiple header entries and content data that are omitted in the example
above. Beside a wide range of data type representations that are not used in the example
above, the framework distinguishes between strings, delimiters and static values. Whereas
static values are not changed at all, strings indicate values that can be changed during the
fuzzing procedure. Delimiters are values that affect the inner structure of the request and
are expected by the program to be tested (for example: the HTTP host header ("Host")
and its content ("webapphost.com") are separated by a colon). It can be configured whether
delimiter values shall also be changed during the fuzzing process.

The following listing shows an HTTP request as it is generated when executing the fuzzing
description above:

```
HEAD %01%02%03%04%0a%0d%0aADSF/directory/anysite.jsp HTTP/1.0
Host: webapphost.com
```
As the framework is configured to only make substitutions of blocks with values of a predefined list, its behavior is deterministic and reproducible. The description above for example generates 8859 different requests before all fuzz paths are exhausted.

The following descriptions of HTTP requests have been modeled and were used for the testing efforts:

- HTTP Basic Authentication
- HTTP GET
- HTTP HEAD
- HTTP POST (formdata)
- HTTP POST (urlencoded)
- HTTP SOAP

All descriptions have been used to send data to the web server using the web application firewalls as reverse proxies. Therefore the web application firewalls had to process the malformed requests.

### 4.5.2 Results

All web application firewalls have been tested using all developed descriptions during a period of three weeks. In connection with phion airlock, Breach Security ModSecurity and F5 Networks BIG-IP ASM no implementation flaws in the parsing routines could be detected. As far as Artofdefence Hyperguard is concerned, a denial of service vulnerability could be found.

The vulnerability was triggered by the test cases 3465 to 3470 of the description for HTTP POST (formdata). The test cases do not lead to an immediate crash of the system but rather in a high system load as far as CPU and memory usage are concerned resulting in repeatedly unanswered requests in the range of the aforementioned test cases. To demonstrate the causes of the vulnerability, the HTTP request generate by test case 3465 is shown in the following listing:
As can be seen, the POST request sends form contents using a valid multipart/form-data encoding. The abnormality in the shown request can be found in the Content-Length header which is set to an unreasonably high value not representing the length of the data actually sent. As far as could be found out without access to the source code of the implementation, the length value is used to allocate memory on the system. Simultaneously the requests leads to a high CPU load if sent repeatedly. It could be found out that child processes serving the requests (and allocating the high amount of memory) are not immediately killed after the (per se too short) request was finished but persist for several seconds.

By choosing a high Content-Length number and send repeated requests an attacker is therefore able to consume significant system resources. A denial of service cannot be achieved with a single request (as long as the attacked system has enough RAM) because Artofdefence Hyperguard works as a module for the Apache web server which discards requests with a too high length (depending on the configuration). The vulnerability can be used to provoke a kernel panic as the values for free RAM and free SWAP space steadily decrease to zero. Afterwards the system has to be rebooted in order to be functional again.

The vulnerability was reported to the vendor on the 22nd of May 2009 using the bug tracking system. An updated version of the product is now available.
4.6 Conduction of penetration tests

The following chapter covers the results of penetration tests that have been conducted on the web application firewall administration interfaces. Please note that the tests were only of limited scope as they were not the main objective of the project. The tests here only cover the administrative functions of the products that normally are only available to administrators (management interfaces).

While testing the administrative interfaces it could be found out that they are not covered by the same ruleset that is applied to the web applications to be protected. In general that enables to exploit found vulnerabilities more easily than with an additional protective ruleset. The following URL demonstrates a cross-site scripting flaw in the management interface of the tested version of Hyperguard (already fixed in the current release):

1 https://10.25.99.12:8082/adminserver/python/gwtguiserver.py/getDebug?sessioni...%3Chtml%3E%3Cbody%20onload%3Dalert('xss')%3Ed=1

Figure 4 shows the vulnerability that can for example be used to steal cookies or to phish login data with the help of an unaware user. The vulnerability only affects users of Microsofts Internet Explorer 7 because the browser parses documents where a closing html- and body-tag is missing. Other browsers do not parse such documents. If the closing tags are included, the web application firewall masks the brackets and therefore stops the attack. However, there is a second cross-site scripting vulnerability in the Hyperguard management interface that is interpreted by all browsers. The vulnerability is triggered when a new user is added by an administrator. If the username contains script code, the value is printed to the user list without filtering. The impact of the vulnerability is considered low because it can only be exploited by a user that already has administrative privileges. Nevertheless it enables an attacker to steal accounts of other administrative users with possibly higher privileges (e.g. by stealing the cookies using the cross-site scripting vulnerability).

The vulnerabilities have been reported to the vendor of Hyperguard and are already fixed in the current release.

The management interface of the F5 Networks BIG-IP web application firewall is also prone to a cross-site scripting vulnerability. The affected function is used to display error messages in case a request to the administration interface cannot be served successfully. The
error message displayed to the user is not escaped properly, enabling an attacker to insert arbitrary script code. The vulnerability can be demonstrated by accessing the following URL:

```
https://192.168.11.13/dms/login.php?msg_id=<script>alert(1)</script>
```

Figure 5 shows that script code is executed in the context of the browser session enabling an attacker to steal cookies, etc. At the time of finding the vulnerability was already known by the vendor and fixed in an updated release.

The web interface of phion airlock is protected by a login that requires a valid username and the according password. All users of the web interface are also system users and are able to log in via SSH for example. The fact that users are stored as system users and given the standard Solaris operation system settings leads to the situation that all passwords are truncated at 8 characters without a specific warning. This enables an attacker to conduct brute force attempts more easily even though success is still unlikely if a good password is chosen.

phion airlock (version 4.1-10.41) is also vulnerable to a remote denial of service attack on the management network interface. This vulnerability affects all protected web servers and applications, because after exploitation the web application firewall cannot handle any further requests and must be restarted manually. In order to conduct the denial of service there is no authentication needed, so the attack can be started by an internal attacker with access to the management network interface or via cross-site request forgery with a single HTTP GET request. The vendor describes the vulnerability as follows:

>>The airlock Configuration Center shows many system monitoring charts to check the system status and history. These images are generated on the fly by a CGI script, and the image size is part of the URL parameter. Unreasonably large values for the width and height parameters will cause excessive resource consumption. Depending on the actual load and the memory available, the system will be out-of-service for some minutes or crash completely, making a reboot necessary.<<

After the initial reporting further research showed that the vulnerability can also be used to execute arbitrary system commands. This allows attackers to run operating system commands.

---

1 https://techzone.phion.com/dos-vulnerability-4.1-sysmon-images
commands under the user of the web server (uid=12359(wwwca) gid=54329(wwwca)). The vulnerability was reported on April 29th, 2009. According exploits will not be published. Both security flaws were addressed by a hotfix and were patched with airlock HF4112. The vulnerabilities are also fixed now within airlock release 4.1-11.18.
5 Conclusion

The general impression of web application firewall technology gained during this project is that web application firewalls can indeed raise the security level of certain vulnerable applications. Nevertheless it must be clearly stated that the additional layer of defense is partly porous and does not replace the secure development and operation of web applications. It also must not be overseen that a web application firewall is an additional device that is placed between the client and the web server and is therefore an additional device that can have influence on the availability of the overall system. It is also an additional system that can have vulnerabilities or other forms of implementation flaws and requires regular maintenance.

Additionally it has been shown that web application firewalls can also be the target of successful attacks (cross-site scripting flaws, cross-site request forgery, denial of service, command execution, etc.).

When defining rules for a specific web application or modifying the standard ruleset it is very important to test the whole web application and all provided functions for their correct functionality. This can for example be done using automated testing frameworks. In the course of the project often certain functionalities of the web applications used for testing have been rendered unfunctional because of predefined rules of the web application firewalls. As unexpected side effects like this can occur with every change of the rules or the web application itself, comprehensive testing is necessary.

The use of web application firewalls can generally be recommended for virtual patching purposes. That means that between the emerging of a new and previously unknown vulnerability and the deployment of the new and tested release possible attacks to the vulnerable application can be blocked by the web application firewall. That also gives developers and testers more time to develop a source code patch while the vulnerability is virtually patched in the meantime. Additionally, web application firewalls can also provide a baseline protection even if certain vulnerabilities of the application to be protected are not (yet) known. An organization using web application firewalls must however be aware that these products cannot cover all vulnerability classes at the same level. The vulnerable test applications developed in the course of this project have been used to determine which classes are covered to which degree.

Whereas vulnerability classes like browser-based attacks, interpreter injection and inclusion of external content have been covered in 60-70% of all cases, other classes like information disclosure or brute force are hardly handled. Whether the reached percentage provides enough protection for a certain application must be decided individually for each case. Generally speaking, the protection level for the three vulnerability classes mentioned above was higher than expected. It is nevertheless advisable to invest in the secure development of web applications and not just in web application firewalls as certain vulnerability classes can hardly be covered or require that vulnerabilities of the application to be protected are already known.
References


