Web Application Firewall

Team name - DarkDante
Shilpa Gupta 110948405 (shilgupta@cs.stonybrook.edu)
Bharath GM 110945903 (bguruvayoorm@cs.stonybrook.edu)
Pulkit Sharma 110900867 (pusharma@cs.stonybrook.edu)
Ali Abbas Hussain 110951463 (alhussain@cs.stonybrook.edu)

Table of Contents

Introduction

A web application firewall (WAF) is used to protect web applications against web based attacks such as SQL injection and cross-site scripting (XSS). The WAF sits in-front of web server and monitors all the incoming traffic to the server from the client side before passing it on to the actual server. The WAF protects the applications against any form of attacks in such a way that it detects and blocks any malicious requests, while allowing regular requests to pass through to the web-server.

There are two methods of developing a WAF

- Implement it as an Apache module
- Deploy the WAF as a reverse proxy server

We chose the second approach because Reverse Proxy will give a generic WAF which can be used with not just Apache but any kind of web server, thus our WAF will have a more universal appeal. The WAF protects web apps from two kinds of attacks:

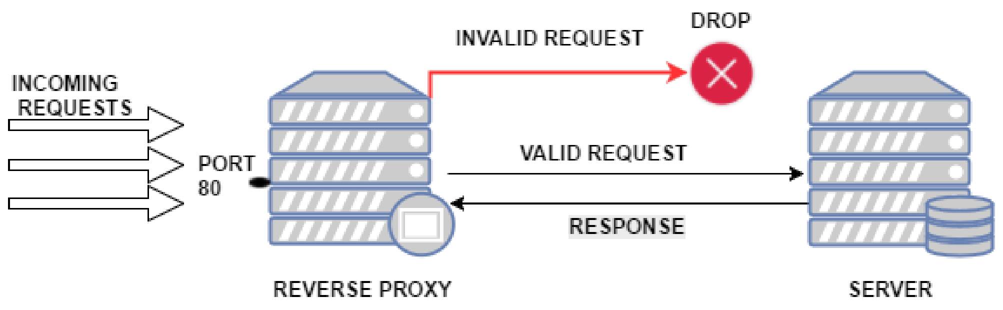
Signatures:

An attack signature is a unique string that is characteristic of malicious intent, it can be used to exploit the vulnerable web application. Our firewall encodes these known signatures and blocks any request in which these signatures are present. The signatures are used to any kind of XSS attacks, any request with <bots> or SQL injection attacks.

Anomaly Detection:

Any known attacks can be prevented using signature detection but attackers can still find a way to intrude into our application using methods that do not have any known signatures. To protect our systems against unknown attacks we use anomaly detection technique which can be trained to learn how proper traffic looks like. If the firewall detects some traffic which deviates from the normal behavior observed, it blocks that request. The techniques involve running the firewall in two modes: ‘train’, ’test’. In the ‘train’ mode we train our firewall to learn about the “legitimate traffic”, we establish some heuristics based on the observation and store it. Once the firewall has gathered enough information we switch from train to ‘test’ mode. In the test mode the firewall looks for the incoming traffic and judges its nature based on the heuristics established, if the firewall detects anything malicious/suspicious, it drops the request and sends and an appropriate message to the client.

Architecture of WAF



Design Description

Tornado

Tornado is a Python web framework and asynchronous networking library. Tornado has the ability to scale to thousands of open connections by using non-blocking network I/O, a good choice for applications that require for long polling, Web Sockets and long-lived connection to each user.

Each request is associated with a port, and a connection is established through which the data reaches the reverse proxy server. The reverse proxy then validates the request and transfers it to the actual server if found valid.

Memcache

This is an open source caching system, intended for use in speeding up dynamic web applications by alleviating database load. It ensures high-performance, and has a distributed memory object. Memcache is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Modes of operation

The firewall can operate in three modes: training, test and reset. The mode of operation should be passed as a command line argument while running the WAF. The syntax for starting the WAF is:

```
sudo python proxyServer.py <mode>
```

Where mode can be :

start_train : Training mode
start_test : Testing mode
reset : Reset the data collected

The working of each of the modes are explained as follows:

Training mode

In this mode, the incoming requests are used for training purpose. The signature validation step, which verifies the content for attacks, is present in both training and testing phase. The request is used for training only after ensuring that the content is valid (it does not contain any blacklisted word/expression). Training happens for the entire web site and also per individual page basis. For the entire website, the maximum number of parameters seen in any request across the entire website is kept track. For each specific parameter seen in a page, the following information is recorded:

- Total count: Keeps track of how many times request has come to the page (used for calculation of average and standard deviation)
- Length: The length of the parameter in the request
- Character set : The character set for values seen for the particular parameter.

Character set can include numbers, alphabets and symbols. This is implemented as a bitmap, with bit positions from LSB set as follows:

Character set	Bit position from LSB
HAS_AL	1
HAS_NUM	2
HAS_SYMBOL	3

This bitmap takes care of all the combinations between alphabets, numbers and special characters, which can be cross-checked in the testing phase. All the values are updated in the memcache - the cache used to store the trained data in the server side.

Storing values in the cache

Memcache is used to store and retrieve the trained data in a fast and efficient way. The maxParams value stores the maximum number of parameters seen across the entire website. Each webpage URL is made a key in the cache, which stores a dictionary of values that include maxParams per page. Each parameter in the page is represented as a key in the dictionary, which contains count, length and characterSet.
Example structure stored in memcache:

```
{
  'a/b/c/url1':{   #values tracked keeping the URL as key
    'Param_1':{
      'count' :'3',       # Number of times the parameter is seen in the page
      'length' :'50',     # Total length encountered
      'characterSet' :'4' # Decimal representation of the bitmap
    },
    'Param_2':{
      'count' :' ' ..“,
      'length' :''.. “,
      'characterSet' :''..”
    },
        ‘maxParams’ : ‘ ‘  # Maximum parameters per specific URL
  },
    'a/b/c/url2':{
    'Param_foo':{
      ...
    },
    'Param_bar':{
      ...
    },
        ‘maxParams’ : ‘’
  }

  ‘maxParams’: “ “   # Maximum parameters for entire web site
}
```

Testing mode

Detection Methods :

Signature Validation - Our firewall supports signature validation on headers and request parameter values for GET/POST type requests. We have created a configuration file named **blacklist.py** which consists of a list of regex/strings to detect and prevent. Any http request containing parameter values that match with any of the regex’s will be blocked . By default we have included the following regex’s. User can add/delete regex’s as desired.

Regex for SQL Injection - *\b(?i)(ALTER|CREATE|DELETE|DROP|EXEC(UTE){0,1}|INSERT(+INTO){0,1}|MERGE|SELECT|UPDATE|UNION(+ALL){0,1})\b*

The above regex will block signature attacks containing sql queries in any header or request parameter values such as
Select * from …..
INSERT INTO …..
UNION all select ….

Regex for other attacks - (?i)((<script|script>)|(\b(bot)\b)|(\.\.\/))

This regex will block the following type of attacks -

Basic XSS Attacks - request parameter values containing a <script> or </script> tag.
Directory traversal Attacks - Values containing ../.. Type of strings.
Bot attacks - Parameter values containing “bot” keywords.
User can add his own regex expressions in the list in blacklist.py in order to block such requests.

Anomaly Detection: For a specific page we search our memcache to get the stored values in our memcache and retrieve all the parameters and related information stored during the training phase. Next, we test the incoming request on this criterion.

Maximum number of parameters seen for all requests in “train mode”: we compare the number of parameter for the page with the maximum observed parameters for all pages in the training pages. If the actual number of parameters is greater than this value, the request is dropped.

Maximum number of parameters seen for the specific page: we compare the number of parameter for the page with the maximum observed parameters for that page in the training page. If the actual number of parameter is greater than the observed value, the request is dropped else the control passes to the next check.

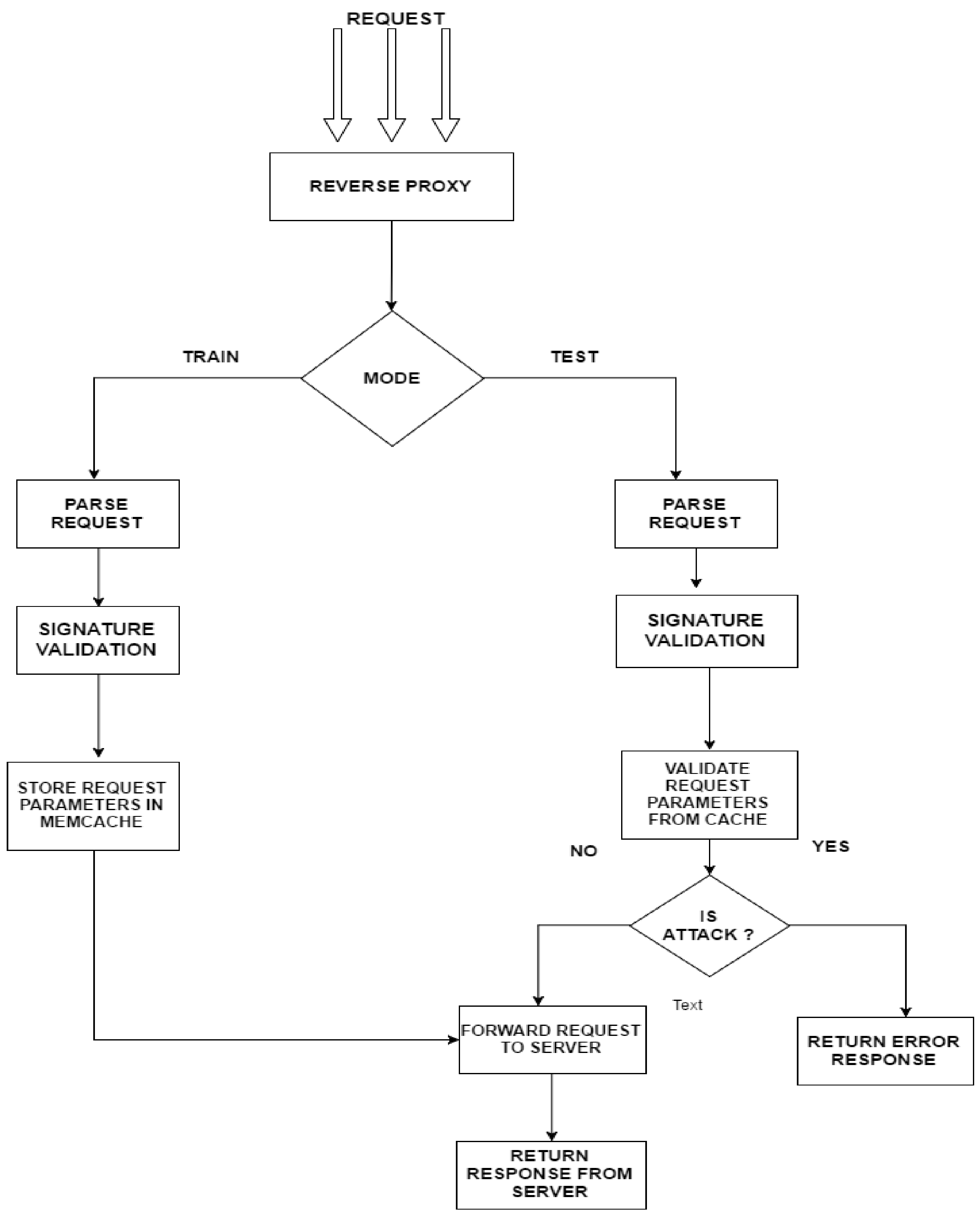
- **Average length of valued for every parameter of specific page:** for each of the parameter we calculate the observed average length by using the stored length and count, now we compare the actual length, if it lies within average +- 3rd standard deviation the checks passes else fails.
- **Character Set of any specific parameter:** checks whether the actual character sets for a parameter of the page is a subset of observed character sets, if check passes the request if forwarded to the server.

If any of the check fails, the request is blocked and an error message is returned to the client. Also, if the entry for a requested page is not present in our memcache database, then only global Maximum number of parameters check is performed. If passed the request is forwarded to the server without any processing.

Reset mode

This will clear the memcache data stored during the training mode.

Flow Chart



REQUEST
REVERSE PROXY
TRAIN
MODE
TEST
PARSE REQUEST
SIGNATURE VALIDATION
STORE REQUEST PARAMETERS IN MEMCACHE
PARSE REQUEST
SIGNATURE VALIDATION
VALIDATE REQUEST PARAMETERS FROM CACHE
NO
YES
IS ATTACK ?
Text
FORWARD REQUEST TO SERVER
RETURN ERROR RESPONSE
RETURN RESPONSE FROM SERVER

How to run

- Preferred OS - Linux Ubuntu
- Packages to be installed -
  Python version - 2.7 - https://www.python.org
  Tornado - “pip install tornado”
  libMemcached- http://libmemcached.org/libMemcached.html
  Pylibmc - “pip install pylibmc”
- Configuring the web application -
  We host our reverse proxy on default port 80. We host our web application on port 8008.
  This can be changed to any ports by configuring Constant.py file of reverse proxy.
  WEB_HOST_PORT for web application port.
  RP_PORT for reverse proxy port.
  Start the web server on port 8008
  This port can be protected from public access via ufw.
  Setting up the reverse proxy.
  Run the file proxyserver.py with one of the following command line arguments -
  start_train - To run the reverse proxy in training mode.
  start_test - To run the reverse proxy in testing mode.
  reset - To reset the training data .
- Start by running in training mode.
- After surfing the website , stop and restart the reverse proxy in testing mode.

Conclusion

Web application firewall using reverse proxy was tested on vulnerable Joomla modules. The WAF collects required information in the training mode. During testing it detects attacks in the headers as well as the parameters, and successfully filters the vulnerable requests. The architecture using reverse proxy makes the firewall scalable and compatible with any server. The firewall can be integrated easily and can be maintained as an independent module as it is loosely coupled with the server.

As the technologies continue to develop and change, firewalls may not be the ultimate solution towards all the attacks. Nevertheless, WAF is an important component in any client-server application that exchanges information. The evolving methods of attacks and increased exploits of vulnerable modules makes firewall an integral component for any customer facing system.

References

https://pypi.python.org/pypi/MapProxy/1.9.0
http://www.tornadoweb.org/en/stable/

https://memcached.org/
https://www.symantec.com/security_response/attacksignatures/
http://securitee.org/teaching/cse509/projects/project1.html