Vyper-CMS™ DDoS Filter

 Saturday, August 08, 2009

Table of Contents

 Saturday, August 08, 2009

What is DDoS ?

According to Wikipedia: Denial-of-service attack

“A denial-of-service attack (DoS attack) or distributed denial-of-service attack (DDoS attack) is an attempt to make a computer resource unavailable to its intended users. Although the means to carry out, motives for, and targets of a DoS attack may vary, it generally consists of the concerted efforts of a person or people to prevent an Internet site or service from functioning efficiently or at all, temporarily or indefinitely. Perpetrators of DoS attacks typically target sites or services hosted on high-profile web servers such as banks, credit card payment gateways, and even root nameservers.”

How to protect against DDoS ?

Request Filtering

Request Filtering is a method of disallowing requests based on a set of criteria.

The criteria can be as simple as the number of requests per unit of time, for the purposes of keeping it simple let’s focus on simply limiting the number of requests from a specific IP address per unit of time.

State Based Request Filtering

State based Request Filtering involves tracking specific IP addresses and the number of requests per unit of time.

Stackless Python

Stackless Python can ease the development process while optimizing the performance of the Request Filtering process.

Individual Tasklets can tack each IP address based on the time of arrival.

Whenever a new IP address arrives a Tasklet search is performed. If no Tasklet is found for the IP address a new Tasklet is created. A message is dispatched to the Tasklet for the IP address with the time of arrival. The Tasklet tracks how many times the IP address arrived per unit of time. If the unit of time is 1 second and the metadata for the system says there should be no more than 1 request per second from each IP address the Tasklet updates the disposition of the Request such that the Django Middleware can handle the Request.

Django Middleware

Django handles the processing for the Web App.

Stackless Python handles the DDoS Request Filtering.

Django Middleware provides the interface between Python and Stackless Python as Django allows the outbound Response to be intercepted and handled by allowing content to be sent back to the requestor or some kind of error condition as would be the case when system detects a DDoS Attack.

Hash Request Filtering

A simpler method of Request Filtering could be one that involves nothing more than a simple Hash or Dictionary where the key is the IP address and the time of arrival is the value – this is the typical way to conceptualize the data.

What happens if the Dictionary (Hash) is flipped inside-out ?

Flipping the Dictionary produces a time based statistical analysis of the IP address arrivals as the key becomes the time period and the value is the a Dictionary of IP address. Now whenever any time period has more than one duplicated IP address the Request is handled as an error rather than as a valid Request.

Data Retention Policy

Data should be retained using a sliding time frame to limit the amount of data in RAM; 30 minutes for Stage 1, 60 minutes for Stage 2 and 90 minutes for Stage 3.

Request Deflection

Django Middleware handles the DDoS detection as the Request enters the system.

Requests that are known to be DDoS Requests are converted into Error Responses before the rest of the Django App knows about the request thus relieving the system of having to perform the typical processing associated with the Request.

DDoS Goals

DDoS seeks to flood the available bandwidth with nonsense requests that respond with some kind of bulky data.

DDoS seeks to bog down the web servers to make them respond more slowly until the web servers are not able to handle the requests and processing comes to a halt.

DDoS seeks to keep legitimate users from getting to the web server and thus deny service to others.

DDoS Protection

Routers can deflect DDoS requests thus keeping them from hitting the web servers.

The web servers can deflect DDoS requests by sending back a 301 response.

301 Response

A 301 Response tells the requestor the document has been permanently moved to a new location or URL.

The URL for the 301 need not exist and can be completely fake for the purposes of telling the requestor who may be involved with a DDoS about the response.

Anyone who codes a DDoS would have to handle redirection responses or run the risk of losing a chance to hit the end server and thus the DDoS fails.

Use of a 301 Redirection as the Response to an assumed DDoS attacker limits the data payload and thus limits bandwidth consumption while telling the attacker to follow a path that dead-ends with nothing but a failed request.

Escalating Request Deflection

Use of a Threshold based Request Deflection scheme can handle the cases when a legitimate user may have fallen into having been misclassified as being a DDoS attacker without immediately triggering the 301 Request Deflection.

Stage 1

Stage 1 results in a 301 Response leading to an image that says the site is too busy to handle the request with a note that says try back later.

Stage 1 is triggered by reaching an initial threshold of say 2 requests per second.

Stage 2

Stage 2 results in a 301 Response leading to an image that says the site is going down for maintenance for a period of time.

Stage 2 is triggered by reaching an initial threshold of say 5 requests per second.

Stage 3

Stage 3 results in a 301 Response that leads to a dummy URL leading to nothing but a failed request.

Stage 3 is triggered by reaching an initial threshold of say 10 requests per second.

De-escalating Request Deflection

Once a requestor's IP address has been flagged as DDoS and handled by a Staged 301 Response the goal of the system is to cause the requestor to back-off for an increasing period of time before making another request. This can be accomplished by assuming the DDoS Attacker will not back-off.

Persistence of Stage

Persistence of Stage assumes the DDoS Attacker will not back-off. Each IP address is classified as being Stage 1 faster with each successive request within a 30 minute window. Stage 2 requests tend to get escalated to Stage 3 and Stage 3 results in a failed request.

Stage 1

Stage 1 is assumed to be persistent for 30 minutes.

Stage 2

Stage 2 is assumed to be persistent for 30 minutes.

Stage 3

Stage 3 is assumed to be persistent for 90 minutes.

