New Sensing Technique for Detecting Application Layer DDoS Attacks Targeting Back-end Database Resources

David Beckett, Sakir Sezer, John McCanny
CSIT, Queens University Belfast, Northern Ireland

Abstract—Distributed Denial of Service (DDoS) attacks targeting the application layer are becoming more prevalent due to a lack of suitable defence solutions. Existing research treats the web server environment as a black box, by only monitoring the edge network traffic; however, we believe that this approach limits the accuracy of the detection system as it does not protect the back-end database servers. In this paper we propose a new sensor located within the back-end system, which can produce additional database features. This allows for real-time insight into the actual database workload caused by each user enabling the detection of DDoS attacks targeting high database consumption resources. These resource metrics are analysed in real-time on a live website, using a decision tree classification engine. Our preliminary results show that a low rate asymmetric attack as low as 1 request every 10 seconds can be detected using these proposed features.

Index Terms—Application Layer, DDoS, Sensor, Backend, Back-end, Database, SQL Database, HTTP, Asymmetric, Detection, Boosted Ensemble, Decision Trees

I. INTRODUCTION

Distributed Denial of Service (DDoS) attacks attempt to make a network or server unavailable to its intended users. DDoS attacks targeting web services poses a serious threat to everyday life as we see more and more critical services being provided on the internet. In the Arbor Network infrastructure security report volume X [1], it states that in 2014, 38% of respondents experienced more than 21 attacks per month. There are various motivations behind these attacks, including political and criminal activity, however the motivations behind the attack are often unknown [1]. Regardless the attacks still cause considerable disruption to the organisations involved.

The first widespread DDoS attack was in 2000 when Mafiaboy, a 15 year old hacker, took down Amazon, CNN, Dell, eBay and Yahoo [2]. Historically, DDoS attacks have targeted the network layer, aiming to take the network devices offline by consuming the bandwidth, TCP connections, memory or processing power of the network switches and firewalls.

In recent years, it has been found that the application layer could be attacked using much lower bandwidth. The attacker can cause the application running on the end server to consume large computational resources with relatively low request rates, causing a denial of service. One example is Slowloris, a DDoS tool that targeted a flaw with Apache, a popular web server application [3]. Due to the low bandwidth requirements of Slowloris, anonymous networks such as TOR network could be used to hide the origin of the attacker.

High rate application flood attacks have also been used to consume the server’s resources, the most popular being Low Orbit Ion Cannon (LOIC) which was used by Anonymous to attack Paypal and Visa by flooding it with HTTP GET requests. Attackers may also rent a botnet from an online marketplace, allowing for a DDoS attack to be launched just from a few clicks on their web browser.

Currently, most DDoS attacks are based on the network layer [4]; however, due to the adaptation of Content Delivery Networks (CDN) and deployment of network layer security appliances, the application layer is becoming the new attack front for DDoS.

The latest DDoS attacks are using an asymmetric attack method in which highly computational resources are targeted. The BBC website was successfully attacked on the 31 Dec 2015 by a DDoS attack which affected the back-end database, taking the BBC website offline for three hours [5]. Although in-depth analysis of the attack has not been released, it is likely that the attackers targeted webpages, which required heavy database access. Also in 2016 the Mirai botnet was used to attack websites with HTTP flood attacks of up to 1.7 million requests per second [6] suggesting that botnets are transitioning to application layer attacks.

In this paper we propose a approach for detecting application layer DDoS attacks targeting resources containing large database queries. We monitor the queries going to the database and uniquely trace them to the originating user, allowing us to build a database usage profile for each individual. This allows us to detect users trying to overload the back-end server. This sensing technique is our novel contribution.

II. RELATED WORK

Existing research on the detection of network layer DDoS attacks is ineffective at detecting application layer attacks as these attacks target flaws within the application protocol, which is only present in the payload. This prevents traditional IPS (Intrusion Prevention Systems) from detecting any abnormal behaviour, as the TCP connection is valid. Over the last few years, research concentrating on the application layer has been carried out; however, we believe a suitable detection solution has yet to be found.

Flow rate statistics have been proposed in [7]–[11] to detect an attack based on request rate timings; however, this detection
method assumes the request rate is proportional to the damage inflicted on the server, which this paper will later prove to be over-simplistic.

User browsing behaviour analysis has been proposed using Hidden Markov Models (HMM) by Yi Xie et al. [12]–[15] to detect abnormal behaviour. They assume that an attacker would access random pages during the attack, therefore by monitoring each user’s movement through the website, malicious browsing behaviour can be detected using complex behaviour analysis. C. Xu et al [16] extended this detection method using the random walk model to specifically detect asymmetric attacks.

Hidden decoy links have been proposed by D. Gavrilis et al. [17] as bots may automatically navigate through the websites by clicking on links, Gavrilis creates hidden links which are not visible to a human therefore if they are clicked on, it will trigger a bot alert. G. Oikonomou and J. Mirkovic [18] progressed this research by creating more sophisticated types of hidden resources to decrease the possibility of the bot detecting the decoy.

Active protection methods such as CAPTCHAs are regularly seen in industry; however, they require human interaction and cause irritation to users. With the advancement of image processing [19] it is hard to predict how much longer they will be effective against bots. Researchers have proposed further puzzle methods such as resource consumption puzzles by Dean Drew and Adam Stubblefield [20], which prompts the client to perform a cryptography hash sequence before gaining access to the server’s resources.

Defense by offense was proposed by M. Walfish [21], who avoided the detection stage and instead persuaded each user to fight for their fair allocation of the resources using Speakup. Clients would be encouraged to increase their traffic rate during a DDoS attack hence giving genuine traffic a higher percentage of the available bandwidth and minimising the affect of the DDoS attack.

Pandiyarajan et al. [22] proposed a method of detecting attacks targeting URLs with heavy database queries. They approximated the database usage by counting the number of variables in the GET request; however, there is no guarantee that the variables are used for a database query, and therefore cannot give accurate visibility on the resource’s database usage.

III. CHALLENGES OF DEFENDING AGAINST APPLICATION LAYER DDoS

To successfully defend against a DDoS attack, the following conditions must be met.

- The detection tool must use fewer computational resources to defend the attack than the cost of building more server resources
- Must not adversely affect genuine user’s quality of service, otherwise the system will not be adopted
- Must not allow the attacker to easily circumvent the protection by adjusting their attack behaviour
- Capable of stopping both low and high rate request attacks.

Existing research does not meet the above criteria. Request rate statistics cannot detect slow rate request attacks targeting high value targets, this research assumes that the request rate is proportionate to the damage caused which is not the case for asymmetric attacks as will be later shown in this paper.

User browsing behaviour analysis requires large computational power which will be unable to cope with the large level of traffic present in a DDoS situation and it also will not cope with a user browsing the website through multiple tabs or users browsing behind a shared NAT IP address.

Hidden decoy links can be avoided by an attacker by manually creating a viable request flow, software puzzles affect genuine users quality of service as there are a wide range of hardware devices connected online with differing capabilities, also the scheme can also be easily attacked using purpose designed ASIC hardware [23]. Lastly defence by offence affects genuine users quality of service which is unpopular for e-commerce services.

Further innovation involving the defence of DDoS attacks is required which can accomplish the above criteria otherwise DDoS attacks will remain a major threat to society. Existing research treats the server environment as a black box by not monitoring the inner behaviour of the system therefore the resources causing the blockage can not be easily seen. There has been little research involving the generation of novel DDoS sensors, instead current research focuses on the training of classifiers using the limited properties available.

DDoS-shield [9] proposed a very basic form of measuring the servers CPU usage by estimating a workload profile for each resource, however their method was too simplistic as for dynamic resources the workload will vary depending on the complexity of their query or user profile. It will also be difficult to scale to large websites consisting of thousands of pages as each resource would need to be continually individually accessed.

A. Contributions of this paper

We propose a solution which allows real-time insight into the actual database workload caused by each user. This will enable the detection of sophisticated DDoS attacks targeting high value database targets. Our main focus for this research is to combat the asymmetric database attack vector which aims to cripple the database servers operating behind the HTTP server.

Our main contribution is the database monitoring sensors located on the web server environment. This monitoring sensor has not been practically demonstrated or modelled by existing research. We also propose a classifier which can accurately distinguish between benign users and the attackers.
IV. Attack Model

Miu et al [24] showed that DDoS attacks were more effective whilst in the TCP ESTABLISHED state as the web server could be directly attacked. While in this state, the attacker can flood the server with HTTP GET webpage requests or corrupted HTTP POST packets in order to attack the server’s resources.

HTTP GET attacks are commonly split into three categories:

- **Request Flooding Attacks** - The attacker sends requests at a high rate
- **Asymmetric Workload attacks** - The attacker sends requests that require a computationally expensive response from the server
- **Repeated one shot** - The attacker sends requests over a large number of sessions

A. Asymmetric Attacks

A sophisticated attacker will target the elements of the service which will cause the most harm. The load of each website resource varies depending on the computation and memory lookups required to return a response to the user. These computation and memory lookups may be carried out on the same host as the web server, or on a separate server dedicated to that particular service. In the popular OpenCart e-commerce software [25], the website interface’s code is stored on the web server, whilst all of the website data is stored on a MySQL database application which performs the queries required to display the website content. Each of these queries will cause computational strain on the server and it can be seen in Fig. 1 that within the OpenCart website there is vast variation in the database computation required to carry out each of the resources functionality. A sophisticated attacker aiming to carry out an asymmetric HTTP attack would target these high value resources overloading the database server.

The resources at risk for this system are:

- **Database CPU Power** - Database processing power consumed by querying the data stores
- **Database/Webserver Connections** - Each time a database connection is opened, a TCP connection between the web and database server will be established, and the number of network connections available decreases
- **Webserver Threads** - If the database server is put under strain, the response time for the queries will increase which will cause web server threads to remain open for longer; this eventually causes a backlog in the request pool and exhausts the web server’s threadpool; preventing it from accepting new requests

To understand the threat posed by an attack targeting asymmetric resources, a test bed running OpenCart was developed and attacked. The most valuable asymmetric target for the attack was chosen using Fig 1, which was the search interface as it requires a high query count and requires the largest database processing time. The least vulnerable resource to attack would be a static web page, not requiring a database connection or any computation, therefore a static text HTML document was placed onto the same server.

It was found that when a 4000 requests/sec attack was placed against the static HTML page for 1 minute, the website’s response time was 1.2 seconds, showing negligible slowdown. In contrast, when an attack of 38 requests/sec was performed against the website’s search engine, the website’s response time slowed to 67 seconds. This was not due to the webserver becoming overloaded, instead it was the back-end database server that was not able to cope with the traffic. This shows the benefits of targeting asymmetric resources, as the search attack required only 1% of the attack frequency of the static page attack yet caused significantly more damage.

V. System Design

As previously discussed, existing research cannot effectively detect low rate asymmetric attacks due to the limited characteristics available. Our research aims to provide a solution to this issue by generating new features using a novel resource consumption sensor located on the server which will give more in-depth insight into the operations of the system. Using the sensor produced features, a database resource usage profile is created for each unique IP address which will allow a classifier to detect DDoS attackers.

A. Sensor Data Acquisition

The sensor will monitor the load on the database server for each user. The database server has no method of relating each query to the originating website user as it only receives the web server’s IP address as shown in Fig 2, therefore the sensor must be embedded onto the web server’s database module so
that the sensor will have access to both the query and the website user’s IP address.

As the modification is carried out on the web server software, no website content needs to be adapted, allowing the sensor to be installed with ease onto live systems. The sensor will log when a database is opened and closed and when a query or commit is performed. A commit is defined as a SQL command which updates or inserts data into the database. The time taken for the database to respond to a query or commit is also recorded so that the load on the server can be estimated. A summary of the logs produced can be seen in Table I.

![Data Centre Schematic](image)

**Fig. 2. Data Centre Schematic**

Apache is the most common web server application [26] and has been chosen for this research. The Zend PHP Framework has been used to provide the user with a dynamic web interface and provide a connection to a back end database server. This design however should be replicable other web server technologies.

**B. Classification System**

To prevent an attacker from consuming the database resources, a classifier will be used to detect misuse. This could also be implemented using a simple resource quota, however DDoS could still occur if the attacker keeps below these limits but instead scales up the number of bots, hence the required need for a classifier to detect unusual database query behaviour.

1) **Classifier Parameters:** The data collected from the sensors will be pre-processed into features before being passed into the classifier. The features created from the data are calculated in timeslots of 30 seconds and are the following:

- Number of Databases Opened
- Number of Databases Closed
- Number of Database Queries
- Number of Database Commits
- Total Database Query Time
- Average Query Time per Database Open
- Average Number of Queries per Database Open

It is common for a website to only open a database once per resource and close it after it has finished processing the page, therefore the database open count also represents the number of http page requests.

2) **Classification Algorithm:** In order to validate the proposed DDoS sensor features, a classification algorithm has been used. The classifier chosen for this was the well known and popularly used decision tree classifier due to the relative ease of separation of the produced dataset and it’s relatively quick training time [27]. The trained classifier also has the inherent benefit of being constructed using standard logic statements, allowing for the detection system to be easy implemented onto a live system. The trained classifier can also be visually analysed as can be seen in Fig 3, this helps ensure that the classifier is not overfitted. The dataset used in this research are labelled using a binary class, 0 for benign and 1 for malicious. The decision tree recursively splits this dataset into M regions during the training stage using a greedy algorithm. Characterisation of the model is carried out by splitting the dataset using one nominal feature, and by using the information gain as the splitting criterion. The grouping boundaries can later be outputted and represented using a logic tree structure [28] [29]

\[ f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \]

To reduce the complexity of the trained tree and to reduce the possibility of overfitting, the tree is post-pruned to remove branches which provide little classification power. The classifier can be trained with more learners to improve accuracy; however, this requires a longer training time.

To increase the overall classification accuracy and to reduce the possible of overfitting, an ensemble of decision tree classifiers is used. Popular ensemble methods include boosting and bagging. A boosting ensemble works by weighting the data in the training set. By adjusting the weights after each training phase, training records which are difficult to classify can be given more influence in the learning process, hence improving the performance of the hypothesis.
VI. EXPERIMENTAL EVALUATION

A. Test Bed Architecture

The experiment was carried out on a private cloud environment running XenServer to replicate a typical industry environment. Two VMs were created on separate Xen hosts and were given Ubuntu 14.04 as the Operating System with 3.5GHz Dual Core and 2GB RAM. One of the servers was used to host the MySQL database whilst the other server had the Apache web server which ran OpenCart to simulate an e-commerce website.

B. Dataset Generation

To ensure that the system was realistic, the OpenCart database used in the test bed was obtained from a popular operational live shop. All datasets were produced against a website running this operational database.

1) Benign Simulation: To ensure that the test bed is tested using realistic normal load conditions, 7 days of real traffic logs from the operational website were replayed against the test server to generate the database features.

2) Attack Simulation: To determine the best attack pattern against the website, each website resource was analysed to determine its impact on the database and the search resource as shown in Fig 1 was the most valuable target. As previously shown in the attack model section, sending 38 requests/seconds against the search resource managed to put enough strain on the server to cause a DDoS situation. In a normal situation there would also be benign traffic causing a base load on the server therefore only a lower attack rate of 36 requests/second is required.

In a typical DDoS attack a botnet is used. J. Santanna et al [30] researched the structure of active botnets and showed that attacks targeting DNS servers consisted of thousands of nodes, however due to it being an amplification attack the actual bots used can only be estimated but it would at least be hundreds. For our testbed simulation, an attack botnet was created using 360 unique IP addresses, each bot sending 1 request every 10 seconds mimicking the typical request rate of a benign user. To ensure that the behaviour of the benign users during the DDoS attack were taken into account, a subset of 150 of the benign dataset users were replayed simultaneously against the server during the attack. Benign users are required to be present during the attack to ensure full classifier training, as query response times for benign users will be larger during an attack due to database locking. As can be seen in Fig 4, the malicious HTTP Requests (class 1) blended in with the request rates of the benign users (class 0).

The response times of the test bed during the attack can be seen in Fig 5. These response times were obtained by performing two curl commands simultaneously against an HTML document (requiring no database connectivity) and one of the shop’s product pages (database connectivity). Both times where recorded to give analysis of how the servers responded during the attack.
The benign users were present at time 0 whilst the botnet started the attack at 20 seconds (position 1) for an attack duration of 300 seconds, ending their attack at 320 seconds (position 3). It can clearly be seen by the stability of the initial response times of the HTML resource (no database required) that it was not the web server but the database server that was affected by the DDoS attack as only dynamic pages where affected initially, however the backlog caused by the slow database responses gradually consumed all of the overall available web server connections which finally resulted in all server resources to suffer from DDoS from 100 seconds (position 2) onwards.

C. Decision Tree Classification

The decision tree classifiers was trained using the 7 days benign dataset and the dataset obtained during the attack. The raw dataset along with two pre-processed features (Average Query Count and Average Query Time) were fed into the classification system. As can be seen in Figs 6, 7 and 8 there is a discernible difference between malicious users which were labelled with 1 and benign users which were labelled as 0.

With a basic complex tree classifier, the system had an overall accuracy of 97.4%. To increase the overall accuracy, Bagged, Boosted, LogitBoost, GentleBoost and RUSBoost ensemble classifiers using cross validation were trained against the dataset. The Boosted Trees classifier was chosen due it having the best accuracy whilst using a low numbers of learners. A lower number of learners will require less computational power in the training of the classifier; therefore the optimum value of 20 was decided upon. On the training machine (Intel Xeon 4 Core, 8 Threads @ 3.70 GHz) using 20 learners, the classifier was trained in under 3.3 seconds.

Table II shows the accuracy of the boosted ensemble classifier for timeslots of 30 seconds. There is a relatively low overall false detection rate of 2.1% which results in a small number of benign users being affected by a defence mechanism. The boosted ensemble tree classifier gave an overall accuracy of 97.9% for this attack type, a false negative of 2.0% and a false positive of 2.2%.

As the system was attacked using 1 request per 10 seconds, even if the attacker instead wished to perform a DDoS at a higher attack frequency, the system will also be capable of detecting this as it is still well above the trained threshold.

### TABLE II

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>True Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign (0)</td>
<td>97.8%</td>
</tr>
<tr>
<td>Malicious (1)</td>
<td>2.2%</td>
</tr>
<tr>
<td>Benign (0)</td>
<td>2.0%</td>
</tr>
<tr>
<td>Malicious (1)</td>
<td>98.0%</td>
</tr>
</tbody>
</table>
Current application layer DDoS research has become stale due to the lack of innovation involving the generation of new features, and as such successful defence mechanisms for asymmetric application layer DDoS attacks have not been achieved. In this paper, we brought new properties and characteristics to the research field by developing a new sensor which can monitor the database server’s behaviour for each of the website’s users. These new features enable the detection of slow rate asymmetric attacks targeting database resources.

Using a boosted ensemble tree, we have shown that using these new features, our detection system can detect sophisticated DDoS attackers targeting large database resources even at low request rates of 1 website request every 10 seconds. The experimental results presented validates the proposed feature extraction method and provides a robust feature set for the detection of low rate asymmetric database DDoS attacks. The detection tool developed also uses very little computational power to train and does not introduce any additional resource strain on the web server during normal use or during the attack which makes it a practical and effective defence mechanism.

Our research assumes users are shoppers, however administrative users may be present on a system, who may generate large legitimate workloads. To minimise the risk of false positives, we envisage the mitigation system will only be enabled for authorised high consumption administrative users.

It is anticipated that the proposed approach may force the attacker to shift to mimicking a benign user by attacking requests with lower database usage. This attack however will require larger request rates to DDoS the system, hence turning the attack into a HTTP Flood which is easily detected by existing detection methods.

Our next aim is to complement existing DDoS request rate methods and to provide an extended detection approach, advancing detection accuracy and robustness of known application layer DDoS attacks including attacks targeting webservers computational resources and complex regexes. The research produced in this paper significantly extends the feature sets available for the detection of application layer DDoS attacks.

REFERENCES