DDoS Attacks at the Application Layer: Challenges and Research Perspectives for Safeguarding Web Applications

Amit Praseed and P. Santhi Thilagam

***Abstract*—Distributed denial of service (DDoS) attacks are some of the most devastating attacks against Web applications. A large number of these attacks aim to exhaust the network bandwidth of the server, and are called network layer DDoS attacks. They are volumetric attacks and rely on a large volume of network layer packets to throttle the bandwidth. However, as time passed, network infrastructure became more robust and defenses against network layer attacks also became more advanced. Recently, DDoS attacks have started targeting the application layer. Unlike network layer attacks, these attacks can be carried out with a relatively low attack volume. They also utilize legitimate application layer requests, which makes it difficult for existing defense mechanisms to detect them. These attacks target a wide variety of resources at the application layer and can bring a server down much faster, and with much more stealth, than network layer DDoS attacks. Over the past decade, research on application layer DDoS attacks has focused on a few classes of these attacks. This paper attempts to explore the entire spectrum of application layer DDoS attacks using critical features that aid in understanding how these attacks can be executed. defense mechanisms against the different classes of attacks are also discussed with special emphasis on the features that aid in the detection of different classes of attacks. Such a discussion is expected to help researchers understand why a particular group of features are useful in detecting a particular class of attacks.**

***Index Terms*—Application layer, DDoS, attacks, defenses, denial of service, taxonomy, detection, Web applications.**

I. Introduction

WEB APPLICATIONS have made information and services available to users without time or space constraints. People can now browse for information, connect with friends, buy and sell things and perform financial transactions at the comfort of their homes. Large scale e-commerce companies have capitalized on this trend and have made efforts to make their services available online. Not just competitive businesses, governments of a large number of countries have also extended their services online. With Web applications powering a major share of businesses and services, it becomes crucial for organizations, business owners and individual governments to ensure that their websites (and by extension, their services) remain available to users all the time.

As Web applications become increasingly important for businesses and financial institutions, they also become targets for malicious users. A large number of attackers are motivated by financial gain, because Web applications are storehouses of critical, personal information like credit card numbers. However, attacks on Web applications also stem from business competition, policy disagreements as well as political and social issues. These attacks attempt to disrupt the services of a Web application so that its services are unavailable to the users, which leads to a loss of revenue for the organization. A single minute of downtime can cost the organization up to $22000 in revenue [1]. Even more devastating is the loss of user trust or a decline in brand value for the organization. Users will not be interested in the services offered by the organization if periodic outages leave the website unusable. These type of attacks, aptly named Denial of Service (DoS) Attacks, are some of the oldest attacks known but still continue to be a major threat due to the way they evolve and grow.

In the 18 months from January 2015 to June 2016, Arbor networks reports tracking around 1,24,000 DDoS attacks every week [2]. Government websites are often targets of such attacks, and the motivation is usually policy disagreements between the government and the attackers. The global hacking collective Anonymous has launched DDoS attacks against a number of government websites, most recently against the Spanish government in support of Catalan independence [3]. The governments of USA [4], Ireland [5], India [6] and Brazil [7] have also been on the receiving end of such attacks in the past couple of years. These attacks expose the inherent weaknesses in the government infrastructure and the lack of security measures in place. Banking and e-commerce sites are also prime targets for DDoS attacks. Attacks on banking sites can effectively cripple the economy by blocking all online transactions. This presents a serious issue at a time when the general public is becoming more and more inclined towards buying and selling online. A number of U.S. based banks were targeted by DDoS attacks in 2012 [11] and customers were unable to perform transactions for hours. A similar attack hit HSBC bank in the U.K. in January 2016 [10]. Bitcoin websites have also been targeted in the same light as banking websites [9], [16], often calling into question the feasibility of a currency with no physical existence.

Manuscript received January 2, 2018; revised June 28, 2018; accepted August 23, 2018. Date of publication September 17, 2018; date of current version February 22, 2019. This work was supported by the Ministry of Electronics and Information Technology (MeitY), Government of India, and it is part of the Research and Development Project entitled "Development of Tool for Detection of Application Layer DDoS Attacks on Web Applications" during the period 2017–2019. *(Corresponding author: Amit Praseed.)*

The authors are with the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal 575025, India (e-mail: amitpraseed@gmail.com; santhisocrates@gmail.com).

Digital Object Identifier 10.1109/COMST.2018.2870658

TABLE I
IMPACT OF DDoS ATTACKS

Target	Type of Organization	Year	Impact
Github [8]	Hosting Service Provider	Mar 2018	Github servers were taken offline
Brazilian Sports Ministry Sites [7]	Government websites	2016	Sites for the Rio Olympics and associated sites went down
Spanish Constitutional Court Website [3]	Government website	2017	Several websites were taken down or hacked
Bitcoin Gold [9]	Cryptocurrency server	Oct 2017	Cryptocurrency deals went down
HSBC [10]	Banking website	Jan 2016	Financial transactions were blocked for a long time
Bank of America, Chase, Wells-Fargo, PNC [11]	US based banks	2012	Financial transactions were unavailable to the general public
Dyn [12]	DNS Provider	Oct 2016	Websites like Twitter and Reddit went down
Valtia [13]	Heating Systems Provider	Oct 2016	Temperature adjustment failed in a town in Finland
Trafikverket [14]	Sweden Transport Administration	Oct 2017	Trains were delayed, reservations portals failed to function
Lonestar Cell MTN [15]	Internet Provider in Liberia	Nov 2016	Internet was unavailable in parts of the country

DDoS attacks are perfectly capable of disrupting Internet connectivity for a large number of users, sometimes even in large parts of a country. Attacking and taking down a DNS server leaves a large number of websites in the dark because users become unable to resolve domain names, as evidenced by the attack on Dyn in 2016 [12]. Taking down a part of the network infrastructure can block Internet connectivity as well, particularly if there are no alternate connection paths in the infrastructure. The attacks on the Liberian Internet infrastructure were executed in this manner, and left large portions of the country without Internet [15].

With more and more devices being powered online, it is not just computer systems that become affected by DDoS attacks. Heating systems in Finland [13] and transport systems in Sweden came to a standstill [14] after DDoS attacks rendered the systems inoperable. This is in line with a report from the network security company Corero in 2017 that 51% of critical infrastructure organizations in the U.K. were ignoring the risk of DDoS attacks [17]. Table I gives a summary of some of the critical DDoS attacks in the last decade.

The reason DDoS attacks remain a major threat even after so many years is because they have grown and evolved over the years. The attacks initially relied on using malformed packets or flooding the device with network layer packets. As the infrastructure became more sophisticated and defenses at the network layer became more robust against these attacks, attackers moved on to the application layer. DDoS attacks at the application layer have been on the rise for a few years. The Imperva Incapsula DDoS Threat Landscape Report 2015-16 [18] indicates that nearly half of the DDoS attacks were at the application layer. The complexity of DDoS attacks at the application layer are also expected to grow over time. Application layer attacks present a more sophisticated version of DDoS attacks in the sense that they are much more similar to normal user traffic and hence pose a serious challenge in how they can be identified. The attacks are carried out using legitimate user requests, which rules out the possibility of inspecting a packet to label it as malicious or not. As a result both network layer defenses and some of the existing Web Application Firewalls (WAFs) fail to detect these attacks. The fact that these attacks can be executed using multiple protocols at the application layer, both connection oriented and connectionless, compounds the danger.

Our contributions in this work are:

1) To present an integrated taxonomy of application layer DDoS attacks based on how application layer protocols and features are exploited to execute attacks.
2) To provide a comprehensive review of defenses against application layer DDoS attacks with special emphasis on the features that aid in detecting different types of attacks.

The rest of this paper is organized as follows: Section II provides a background of the area, providing a description of network layer and application layer DDoS attacks. Section III proposes a taxonomy of application DDoS attacks and describes the different classes of attacks. Sections IV to XI discuss the detection mechanisms in the literature today for the different types of attacks, with particular emphasis on the features that can be used to detect the attacks. Section XIII-A gives an overview of tools which aid in detecting a few classes of application layer DDoS attacks and Section XIII-B discusses the different datasets that researchers can use in the area. Section XIV discusses the different metrics which are commonly used in evaluating detection mechanisms.

II. BACKGROUND

A. Denial of Service Attacks

Denial of Service (DoS) attacks are some of the oldest attacks against Web applications. The first reported use of what can be considered as a DoS attack dates back to the late 1990s [19]. Since then they have evolved and grown and have become one of the most common attacks against Web applications. The distinction between a DoS attack and a Distributed DoS (DDoS) attack is in the number of attackers involved. A DoS attack typically implies a small number of attackers, sometimes even a single attacker. DDoS attacks are more massive and can involve hundreds or thousands of attackers. These attackers need not be human attackers, and in most cases there are just a few human attackers. The "attackers" in this scenario refer to the systems that are being controlled by the human attackers. Systems that have been infected by malware and are acting as attackers on behalf of the real attackers are called zombies or bots. Attackers usually employ a large number of such bots to form a botnet.

The objective of a DDoS (or DoS) attack is to make a Web server unavailable to legitimate users trying to access it. This can be accomplished in multiple ways, but the core idea is to exhaust one or more of the resources available with the server. These resources could include CPU or database cycles, memory, socket connections or network bandwidth. Attackers may exploit system weaknesses or protocol weaknesses to do so, or they may simply push the server to its limits by using the features provided by the Web server repeatedly.

B. Application Layer DDoS Attacks

A large number of DDoS attacks target the network bandwidth because of the ease with which it can be exhausted. Attackers simply send a large volume of network packets to the server, effectively exhausting the network bandwidth. Network layer protocols like UDP (User Datagram Protocol) and ICMP (Internet Control Message Protocol) are used for this purpose. As time passed, two things changed. Networks and servers became more robust in identifying network layer DDoS attacks and servers could afford better and increased bandwidth. An enormously large volume of requests could still exhaust the network bandwidth and take down a server, but it became more difficult to do so. The attackers responded by moving up the stack to the transport layer. SSL renegotiation attacks [20] exploited the transport layer, but as time passed servers began to defend against these attacks as well. These attacks had a pattern that could be identified and generalized across platforms, and could be strictly classified as malicious.

In recent years, attackers have moved up the stack one more time giving rise to a new trend called application layer DDoS attacks. These attacks do not aim to throttle the network bandwidth, instead they attempt to exhaust server resources like CPU cycles, database cycles, memory or socket connections. There has been a huge growth in the number of application layer DDoS attacks in the recent years. Imperva Incapsula's Global DDoS Threat Landscape Report 2017 [21] reports that for the fourth quarter in a row, there was a decrease in the number of network layer assaults along with another spike in the number of application layer assaults. Reports by Kaspersky [22] mention the fact that "the cream of cybercriminal communities are now turning to Application Layer DDoS attacks".

Attacks carried out at the application layer have a few subtleties which make them unique and different from other attacks.

- *Legitimate requests:* Application layer DDoS attacks proceed through legitimate HTTP packets. There is virtually no difference between an attack request and a normal request. The only difference resides in intent, and not in content. This makes most network level packet filters and even some application layer firewalls ineffective in detecting these attacks.
- *Low Traffic Volume:* Contrary to network layer DDoS attacks, where the network bandwidth becomes the bottleneck, in application layer DDoS attacks, the server resources become the bottleneck. Because of that, the server can be brought down using comparatively fewer requests, and hence the traffic volume is also low. Most of the existing DDoS detection mechanisms rely on the large traffic volume to identify attack. That approach fails in the case of application layer DDoS attack, making most of the existing DDoS detection mechanisms obsolete.
- *Targeted Strikes:* Application layer DDoS attacks are highly targeted. The attack can be carried out on the CPU, database, memory, or socket connections. An attack which attempts to exhaust one resource will not affect the other resources in any way, but the system as a whole will not be able to function. As a consequence, a defense mechanism for one resource is unlikely to be effective for another resource.
- *Resemblance to Flash Crowds:* A flash crowd is a sudden spike in legitimate user traffic to a website, most often due to some noteworthy event or a mega sale in the case of e-commerce sites. An application layer DDoS attack is often confused with a flash crowd because both of these events are associated with a spike in legitimate HTTP requests to a site. Proper identification of a DDoS attack and a flash crowd is essential for any defense mechanism because flash crowds bring valuable traffic to the website, and it would be detrimental to the website if they were wrongly labeled as DDoS traffic and blocked.

III. Taxonomy of Application Layer DDoS Attacks

A fair amount of work has gone into studying and classifying DDoS attacks at the application layer. One of the earliest surveys in the area belongs to Geneiatakis *et al.* [23]. They presented a review of vulnerabilities in the SIP (Session Initiation Protocol) protocol which could be exploited to launch a DDoS attack. However, the work was restricted to a discussion of the avenues of attack and not the defense mechanisms against these attacks. Works by Armoogum and Mohamudally [24] and Hussain *et al.* [25] further examined the area in more detail. They also examined the different defense mechanisms that have been discussed in literature for SIP based DDoS attacks. Jensen *et al.* [26] presented a detailed discussion of attacks on Web services.

TABLE II
COMPARISON OF EXISTING SURVEYS OF APPLICATION LAYER DDoS ATTACKS

Research Work	**Year**	**Area Covered**	**Limitations**
Geneiatakis et al. [23]	2006	SIP protocol vulnerabilities	Discusses security vulnerabilities leading to DDoS attacks in the SIP architecture, no discussion on defense mechanisms
Durcekova et al. [27]	2012	GET flooding and Slow DDoS attacks	Discusses only a few types of attacks, does not provide an in-depth classification of attacks or defenses
Aamir & Zaidi [28]	2013	Network and application layer attacks and defenses	Limits discussion to flooding attacks, does not provide any classification
Zargar et al. [29]	2013	HTTP floods, reflection attacks, Slow Attacks	Detection mechanisms are classified based on deployment location and time of action, does not consider asymmetric attacks the way previous works do
Cambiaso et al. [30]	2013	Slow DDoS Attacks	Focuses discussion of primarily slow DDoS attacks, but refers to any application layer DDoS as slow, does not discuss any defensive mechanisms
N. S. Vadlamani [31]	2013	HTTP floods and asymmetric attacks	Discusses only attacks using HTTP protocol, but neglects Slow DDoS attacks, also does not present a taxonomy of attacks or defenses
Armoogum et al. [24]	2014	SIP based DDoS attacks	Discusses only SIP based DDoS attacks, discusses defense mechanisms but does not present a taxonomy
Hussain et al. [25]	2015	SIP based DDoS attacks	Considers only SIP based attacks and their defense mechanisms
Wang et al. [32]	2015	Application layer attacks and defense mechanisms	Restricts the discussion to HTTP attacks, defense mechanisms are classified only based on deployment location
Mantas et al. [33]	2015	Application layer attacks	Does not examine any defensive mechanisms
Singh et al. [34]	2017	GET flooding attacks	Limits the discussion to only GET flooding attacks and ignores other application layer attacks

They also provided details of how the SOAP (Simple Object Access Protocol) protocol could be exploited to execute DDoS attacks. In the recent years, the focus of the research community has been more towards HTTP based DDoS attacks. Aamir and Zaidi [28] and Zargar *et al.* [29] both examined network and application layer attacks. Aamir and Zaidi [28], restricted the discussion to flooding attacks at the application layer. Zargar *et al.* [29] provided a classification of attacks that included reflection attacks, flooding attacks, asymmetric attacks and slow DDoS attacks. The discussion on defense mechanisms focused on where the solution is deployed and the time of action. Wang *et al.* [32] presented a similar classification of attacks as [29], but only classified defense mechanisms based on point of deployment. Vadlamani [31] examined the different HTTP flooding and asymmetric attacks, the different defenses against them, their merits and demerits. Mantas *et al.* [33] provided a detailed taxonomy of application layer DDoS attacks based on target level, exploit type, attack methodology, attack volume and attack workload. However they did not discuss any defense mechanisms against these attacks. Singh *et al.* [34] provided a detailed taxonomy of GET flooding attacks based on a number of factors. They also examined the detection features that can aid in detecting these attacks. Durcekova *et al.* [27] presented a review of application layer DDoS attacks, but restricted the discussion to GET flooding attacks, and slow DDoS attacks. They also discussed a brief classification of defensive tactics against these attacks. Cambiaso *et al.* [35] presented a taxonomy of slow DDoS attacks, but did not present any discussion about the defenses. A comparison of the recent attempts at classifying application layer DDoS attacks and their defensive mechanisms is given in Table II.

Our work is different from the existing literature reviews in the following points.

- We provide a detailed survey of application layer DDoS attacks exploiting four major application layer protocols - HTTP, SOAP, DNS and SIP.
- We present a taxonomy of application layer DDoS attacks which integrates attacks exploiting the four major application layer protocols.
- We discuss the different features that can be used to detect different classes of application layer DDoS attacks, and discuss existing research works that utilize these features for attack detection.

In this work, application layer DDoS attacks have been classified on the following features:

- *Nature of Exploitation:* This feature analyzes how the exploit is carried out by the attacker. Attackers can find weaknesses in a system to launch DDoS attacks, or they may exploit features in the underlying protocol. In the absence of these weaknesses, attackers can simply exploit features provided by the Web application to launch the attack.
- *Protocol:* Application layer DDoS attacks have been observed mainly using four different layer 7 protocols - HTTP, SOAP, DNS and SIP.
- *Payload Delivery:* This feature focuses on how the attacker delivers his attack payload to the target system. An attacker could directly send the attack requests to the target server, in which case the attack is said to be a direct attack. For application layer protocols which do not rely on a connection (eg. DNS), an attacker can request for a large amount of information from some server, spoofing the victim's address. This launches a stream of messages

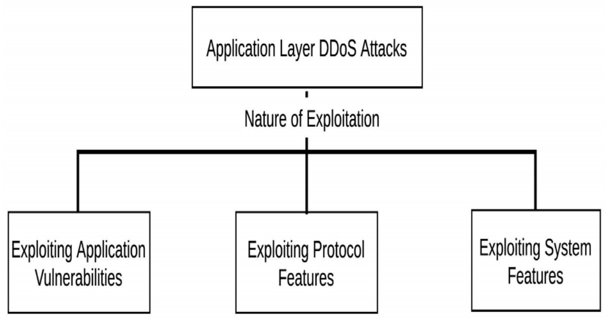

Application Layer DDoS Attacks
Nature of Exploitation
Exploiting Application Vulnerabilities
Exploiting Protocol Features
Exploiting System Features

Fig. 1. Classification of Application Layer DDoS attacks based on Nature of Exploitation.

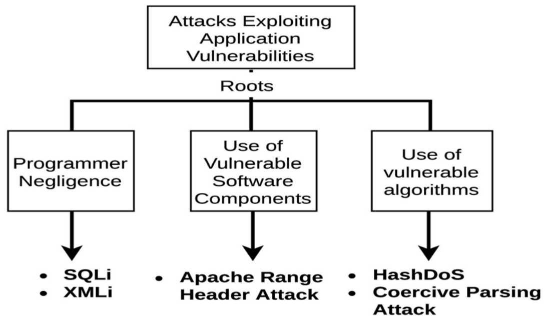

Attacks Exploiting Application Vulnerabilities
Roots
Programmer Negligence
Use of Vulnerable Software Components
Use of vulnerable algorithms
• SQLi
• XMLi
• Apache Range Header Attack
• HashDoS
• Coercive Parsing Attack

Fig. 2. Taxonomy of Application Layer DDoS attacks exploiting System Weaknesses.

to the victim which can effectively knock it offline. This mode of attack is called reflected mode of attack.

- *Overhead of Attack:* Attackers have to expend some amount of resources in order to launch an attack. If the amount of overhead incurred by the attacker is proportional to the damage intended on the victim, the attack is said to be symmetric. However, using specially crafted requests, the attacker can reduce the overhead incurred while maintaining, or even increasing, the damage at the victim. Such an attack is termed as an asymmetric attack.

We have focused on those criteria which can help researchers better understand the inner workings of a DDoS attack, and as such provide an understanding on how these attacks can be defended against. Figure 1 gives the first level breakup of application layer DDoS attacks. The attacks have been classified based on the nature of exploitation into three classes. Each of these base classes have been examined in detail in the following subsections. We have chosen the exploitation level as the root feature for the classification tree because this separates the tree into categories with little or no overlap.

A. Exploiting the System Weaknesses

A large number of Web applications have security loopholes that can be exploited to launch attacks. These vulnerabilities can arise due to three reasons:

- Use of vulnerable software components
- Use/reuse of vulnerable algorithms without patching
- Programmer negligence

A schematic diagram classifying attacks that exploit system weaknesses is given in Figure 2.

1) Use of Vulnerable Software Components: No Web application is designed and implemented from scratch. Existing software components, like load balancers and proxies are often used as such and very little code is actually written by the developer. The use of software components with existing vulnerabilities, knowingly or unknowingly, puts the entire Web application at risk. For example, earlier versions of the Apache server suffered from a vulnerability due to which an HTTP message with a large overlapping range header caused a memory exhaustion and crashed the server [36]. This attack is no longer feasible, because newer versions of the server have patched this vulnerability.

2) Use/Reuse of Vulnerable Algorithms: Most Web applications make use of existing algorithms for hashing or parsing input data. A lot of the time though, this code is often reused with little thought with regards to security. Vulnerabilities in algorithms come to light under certain conditions and can result in a system crash if not handled accordingly. HashDoS [37] was an attack that exploited the use of vulnerable hashing algorithms in Web application servers that use hashing to organize POST input parameters. In the best and average cases, insertion, and access of items in a hash table proceed in $O(1)$ complexity, but in case a collision occurs, the hash table degenerates to a linked list with $O(n)$ complexity. An attacker who supplies a crafted input with a large number of POST parameters can successfully cause collisions in a large number of scripting languages. This can cause the CPU to spend a large amount of time working to resolve collisions and cause a denial of service situation [38].

Another example is a coercive parsing attack on XML Web services [26]. At the receiver end, XML messages have to be parsed before they can be processed. Deeply nested XML packets can cause a sharp increase in CPU usage. So to bring down a SOAP server, the attacker simply sends a SOAP message with a large number of opening tags till the server goes down.

```
<soapenv:Envelope xmlns:soapenv="..."
xmlns: soapenc:"...">
<soapenv:Body>
  <x>
    <x>
      <x>
        <x>
            ... contd.
```

This vulnerability exists only in a Document Object Model (DOM) parser. A DOM parser creates an in-memory representation of the incoming SOAP message during processing. This typically increases the message size by a factor from 2 to 30. For large messages, this expansion can result in memory exhaustion as well. A stream based parser like SAX parser does not experience such a vulnerability, because the entire document is never loaded into the memory at a time.

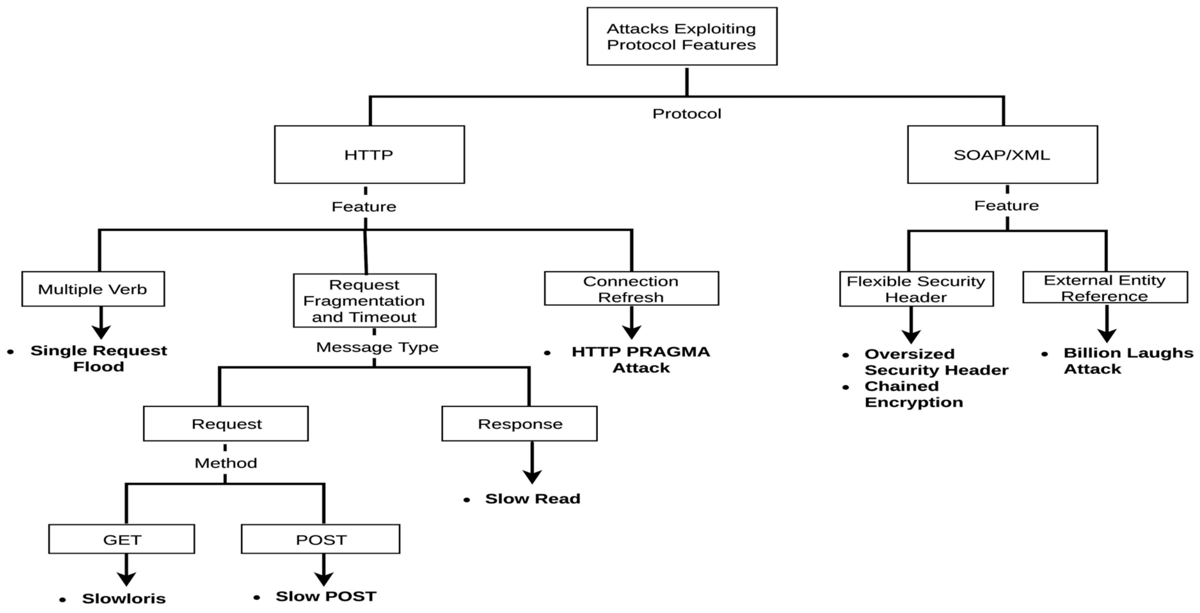


Fig. 3. Taxonomy of Application Layer DDoS attacks exploiting Protocol Features.

3) Attacks Exploiting Programming Negligence: A large number of Web applications operate with an SQL (Structured Query Language) back-end. For these applications, SQL injection is a real threat. SQL injection is usually associated with injecting malicious data or leaking sensitive information. But it can also be used to cause a denial of service situation. All the queries issued by a user make their way to the back-end server and get executed on the database. If an attacker can make the database server perform actions which leads to an inconsistent state or even delete the entire database, the system cannot function and leads to a denial of service. This isn't the most sophisticated way of inflicting a denial of service, but sometimes the simplest attacks are the most brilliant.

The question though is : how to get the database into an inconsistent state? Let us assume the Web application allows the users to search for an employee based on an employee number. The query would normally look like

```
SELECT * FROM employee
WHERE EmpNo = $input$;
```

where *input* is the value entered by the user. Consider the case where the user enters an input like
"1;*DROP TABLE employee*"
The query generated internally is

```
SELECT * FROM employee
WHERE EmpNo = 1;
DROP TABLE employee;
```

Since a ';' in all major SQL implementations acts as a delimiter between queries, the database server will execute both the queries and the table *employee* will be dropped. Such an assault is called a piggybacked query and can be used to cause a denial of service on the application.

B. Exploiting Protocol Features

Protocols are designed to facilitate efficient communication between different parties regardless of differences in bandwidth or computing power. Attackers can use different features of protocols, which were originally meant for efficient communication, to launch attacks and possibly create a denial of service situation. HTTP and SOAP are the two major protocols at the application layer. A classification of those attacks which exploit these protocol features are represented in Figure 3.

1) Exploiting the HTTP Protocol: HTTP was designed to facilitate communication between a human user (using a Web browser) and a Web server. HTTP is a connection oriented protocol based on TCP, which means a TCP connection should be established before communication can proceed. This connection is maintained till the end of communication. Different features of HTTP have often been abused by attackers to launch attacks.

a) Request fragmentation: HTTP was designed with all users in mind, even those with a small bandwidth and hence HTTP allows its users to fragment an HTTP message across multiple packets. An attacker who fragments his HTTP messages into extremely small packets can keep the connection open for an arbitrarily long time. Since Web applications have a predefined limit on socket connections it can maintain simultaneously, an attacker who manages to keep multiple connections open for an infinitely long time can effectively force the server to decline legitimate connections. This class of attacks are called Slow DDoS attacks and come in two varieties based on whether the attack is carried out using an HTTP request or response.

The attacks that make use of an HTTP request are called Slow Write attacks. The most famous slow write attack made an appearance after the 2009 Iranian Presidential elections and is dubbed Slowloris. These attacks were carried out by HTTP connections that sent GET requests to the server in extremely small fragments, sometimes sending individual headers one by one. A normal HTTP GET request is shown here:

```
GET /index.php HTTP/1.1[CRLF]
Pragma: no−cache[CRLF]
Cache−Control: no−cache[CRLF]
Host: testphp.vulnweb.com[CRLF]
Connection: Keep−alive[CRLF]
Accept−Encoding: gzip, deflate[CRLF]
User−Agent: Mozilla/5.0
(Windows NT 6.1; WOW64)
AppleWebKit/537. 36 (KHTML, like Gecko)
Chrome/28.0.1500.63 Safari/537.36[CRLF]
Accept: */*[CRLF][CRLF]
```

HTTP uses a carriage return line feed (CRLF) to denote next line and it uses two CRLF characters to denote a blank line. Typically a blank line denotes the end of the headers in an HTTP request. An attacker can simply omit the double CRLF sequence so that the server assumes the user is not done sending headers. After sending a small fragment of the HTTP headers, the attacker waits. HTTP defines a timeout for all connections, which is the time for which an HTTP connection can remain idle without being torn down. This is not a fixed number but is server dependent. Just before the server timeout duration is reached, the attacker sends the next fragment. This forces the server to maintain the connection. If executed over multiple connections the server runs out of socket connections for legitimate users. The other counterpart of a slow GET attack is a slow POST attack. A GET request has size limitations on it due to the fact that it does not have a body. A POST request on the other hand can be arbitrarily long because it has a request body. The HTTP header specifies a field called *Content−Length* which tells the server how long the message is going to be. The attacker sets the POST request to have an arbitrarily large value of *Content−Length* and then proceeds to send data in small fragments. The server is forced to maintain the connection until either the connection times out or the entire message is received. After a while the server is unable to accept any new incoming connections.

Slow DDoS attacks can also be launched on the HTTP response and are called Slow Read Attacks. The attacker sends a GET or POST request to the server and waits for a response by advertising a small network window. The Web server assumes the user is on a low bandwidth connection and proceeds to send the HTTP response in small fragments. This ties up the connection until the entire data is received. If the attacker can establish multiple connections of this nature, it exhausts the socket connections on the server.

b) Connection refresh: The effect of maintaining a connection for an undefined amount of time can be achieved by using an HTTP header field called PRAGMA [39]. The HTTP PRAGMA header field tells the HTTP server and any intermediate caches that the user wants a fresh copy of the requested resource. This ensures that the request is not satisfied by any of the caches but by the Web server itself, ensuring that the server has to expend processing power. Additionally, whenever an HTTP PRAGMA is issued the timeout for the connection is reset. Use of the PRAGMA field can keep connections open for an indeterminate amount of time, ensuring that socket resources get tied up. It is worth mentioning that the PRAGMA field has no visible purpose in HTTP 1.1, but is still allowed to maintain compatibility.

c) Multiple verbs: HTTP request methods initiate action at the server side, and for this reason they are sometimes called HTTP verbs. Traditionally, a single request contains of a single action or verb. In such a scenario, it is rudimentary that the more workload the attackers want to dump on the server, the more number of requests they have to send. But HTTP has a somewhat lesser known feature which allows the users to pack multiple verbs into a single HTTP request. This means that attackers can compress multiple verbs into a single request and send to the server. The server is forced to perform all the tasks that are requested, which is much more than a normal request. The advantage this presents to the attackers is that the attack volume can be cut down to a large extent, thus helping them evade detection [29], [40].

d) The push for HTTP 2.0: HTTP 1.1, which is currently the standard for communicating with Web servers, is expected to be phased out soon. HTTP 2.0 is poised to replace HTTP 1.1 in the coming years. Most modern browsers and servers do provide options for users to opt for HTTP 2.0. This newer version of HTTP provides a header compression mechanism to reduce the header length, and avoid message overheads. It also allows servers the freedom to push content to the client without an explicit request to save time. It also allows clients to send multiple concurrent requests simultaneously, on a single TCP connection and potentially all within a single packet.

While these features have been designed to improve the speed and efficiency of communication, there have been concerns over security. The feature which allows multiplexing of requests can be used to launch flooding attacks, and even asymmetric attacks. Beckett and Sezer [41] demonstrated that HTTP 2.0 allows an attacker to carry out a much larger attack on the same hardware, with the same packet generation limitation. Imperva also reported that the slow reading vulnerability present in the older version of HTTP was still present in the newer one [42].

2) Exploiting the SOAP Protocol: Just like HTTP is the standard for communication for Web applications, SOAP (Simple Object Access Protocol) is the standard for communication for Web services. Web services involve communication between Web applications without any human interaction. This necessitates the use of a universal language which can be used on all platforms and is easily processed by servers. Readability and ease of use tends to take a back seat because of the lack of human interaction. XML and JSON are the two most commonly used notations which have universal acceptance, and of these XML takes precedence because of its reach. The SOAP protocol is primarily based on XML messages for this reason. The SOAP protocol also has security extensions which enable safe and secure transmission of XML data. However, sometimes the flexibility offered by SOAP allows malicious users to take advantage and force the server to do more work.

a) Flexible use of security header: WS-Security is a security specification which extends SOAP to provide integrity

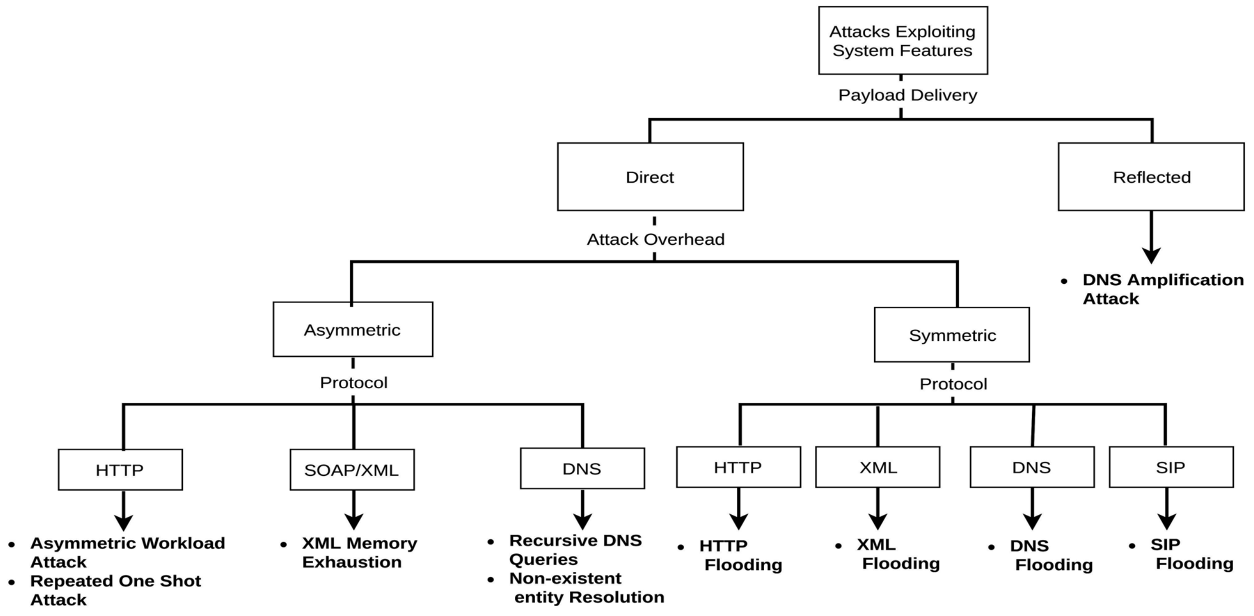


Fig. 4. Taxonomy of Application Layer DDoS attacks exploiting System Features.

and confidentiality of data and specifies the different ways to use XML signatures and encryption to ensure that data remains safe and secure. One of the specifications is the addition of a security header which includes security elements such as decryption keys. WS-Security allows considerable flexibility to encrypt different parts of the message with different keys and possibly encrypt the keys themselves. All the keys must, however, be present in the security header which is encrypted with the receiver's public key. Because the specifications allow flexibility in encrypting the message parts, the security header can be very big, which is allowed by default. However, attackers can purposefully use oversized security headers to crash the system [26]. This is because the security header contains the keys necessary for decryption and must be buffered while the message is processed. If the security header is extremely large the server can potentially run out of memory while processing a few malicious packets. A more dangerous attack may be to encrypt different parts of the document with different keys and then to encrypt the keys recursively. Such an attack destroys the system on two fronts. On one hand the security keys must be buffered while processing the message, and if the attacker uses a deeply nested set of encrypted keys the system can potentially run out of memory. On the other hand, each of these keys should be decrypted before the system can proceed further. Most of the encryption and decryption algorithms are computationally expensive and consume a non-negligible amount of CPU time. Such an attack can thus potentially exhaust the CPU and the memory resources available with the server.

b) External entity reference: XML packets were designed to be compact. However, the XML language allows the users to include a link to point to an external entity, which will be fetched and processed when the XML message is parsed and processed. This is a simple and effective way to maintain the compactness of XML, and at the same time to allow users to send larger entities for processing. However, the same can become a point of exploitation. Attackers can potentially send XML messages with an External Entity Reference pointing to a large document [43]. While the message is parsed, this document is fetched. The entire XML message, of which the document is actually a part must reside in memory for processing to complete, and so with a few simple messages the attackers can ensure the system runs out of memory. A well known attack of this type is called the Billion Laughs Attack [44], and defines nested entities within an XML Data Type Definition (DTD) document to create a memory bomb. This attack causes the parser to generate an abnormally large payload, potentially overloading the application and causing a denial of service.

C. Exploiting System Features

DDoS attacks are simple and easy to launch in the sense that they do not rely on the existence of a flaw or a vulnerability in the Web application. DDoS attacks can exploit a vulnerability or weakness in the server if one exists, but they do not need to. Every Web server has a finite amount of resources at its disposal - CPU, memory, database or socket connections. If attackers can exhaust these resources by sending requests that they do not intend to use, the server has no resources available for serving legitimate demands from normal users. This makes the website unavailable for legitimate users. The most striking fact about these attacks is that even the most secure servers are vulnerable to these attacks. These attacks can proceed through the use of different protocols at the application layer - HTTP, SOAP, DNS or SIP. Of these HTTP and SOAP work with an underlying TCP connection which is connection oriented, while SIP and DNS work with the connectionless UDP protocol. Attacks exploiting server features can be broadly classified into direct and reflected attacks. A detailed taxonomy of this class of attacks is given in Figure 4.

1) Direct Attacks: Direct attacks involve the attacker sending malicious requests directly to the victim server. If the attacker sends a large enough number of requests to the server, the server will be unable to process them and will crash. However, in the process the attacker needs to perform significant work in itself by sending these requests. Such attacks are said to be symmetric. The objective of the attacker is to bring down a server by using as few resources as possible. It is possible to make a server do more work by carefully crafting requests or by sending a particular stream of requests instead of random requests. Such attacks reduce the load on the attacker while maximizing the load on the target server. Such attacks are said to be asymmetric in nature.

a) Symmetric DDoS attacks: Symmetric DDoS attacks are in many ways similar to network layer DDoS attacks. The similarity lies in the fact that like network layer DDoS attacks, symmetric DDoS attacks also work by sending a large number of attack requests to the target Web server, so that it is unable to process legitimate client requests. However, while network layer DDoS attacks utilize network layer protocols like ICMP, attacks at the application layer rely on protocols like HTTP, SOAP, DNS or SIP. A point of difference is that attacks at the application layer do not need to throttle the server bandwidth to cause a denial of service situation. A single HTTP request makes the server perform more work than a packet at the network layer. So the server resources become the new bottleneck in this situation and get exhausted much before the bandwidth of the server gets throttled.

HTTP flooding [45] is the most common application layer DDoS attack which utilizes the HTTP protocol. This is largely due to the ease with which this attack can be executed. The simplest way of executing this attack is to repeatedly send requests to any one URL on the target Web application. More often than not, it is the home page or login page that is attacked. However, a repeated stream of requests to the same URL can be easily identified and blacklisted by the server administrator. So the next line of attack is to continuously send requests to random URLs in the Web application. This works in the same way as the earlier attack, but cannot be detected that easily. Tools like Low Orbit Ion Cannon(LOIC) or other stress testing tools can easily be used to launch an HTTP flood. The fact that these tools can be easily downloaded off the Internet makes it much more dangerous.

SOAP or XML flooding [46] works in the same way as HTTP flooding but is targeted at Web services.

DNS is perhaps the most important protocol in the application layer because it aids in address resolution. A DNS server works by translating a domain name to the corresponding IP address. The fact that DNS operates on the UDP protocol makes it an open target for attacks because anyone can openly query a DNS server at any time. Attackers can easily take down a DNS server by sending a large number of queries to the DNS server. After a point of time, the DNS server cannot accept any new queries and effectively goes offline [47]. This presents a much more serious problem than a single Web server crashing. If a single Web server goes down only that organization is affected. But when a DNS server goes down a large number of users are effectively cut off from a large part of the Internet because the address to IP resolution is the first step in accessing any Web application. The attacks on Dyn DNS service in 2016 are a prime example [12]. When the services of Dyn went down, a large number of popular websites including twitter and Reddit went offline for hours.

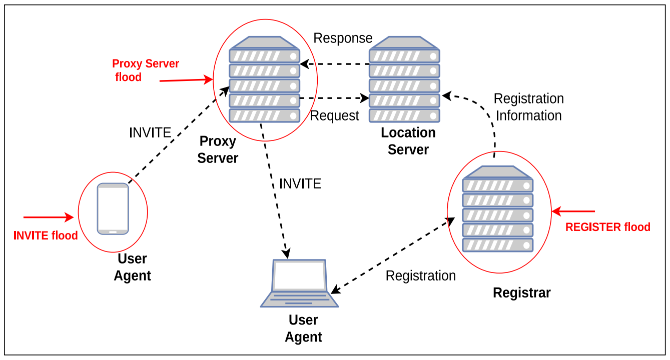


Fig. 5. Avenues of Attack on SIP Architecture.

VoIP (Voice over Internet Protocol) is a new avenue of communication which offers the services provided by a Public Switched Telephone Network over the Internet. Falling data costs make it much more affordable for users to use VoIP as a means of communication nowadays. Session Initiation Protocol (SIP) is the most popular protocol used in VoIP. A SIP architecture consists of a caller, a registrar, a location server, proxy server and the callee. The caller and callee are uniquely identified by their IP and URI. When a caller connects to the network, they register with the registrar server. The location server maintains a record of the location of the user agents by means of their IP. A proxy server is meant to forward the call from the caller to the callee. When a caller places a call to the callee, the caller sends a SIP INVITE message to the callee via the proxy server. The proxy server contacts the callee and at the same time sends back a TRYING message to the caller. When the callee receives the call, it starts ringing. This is indicated by a RINGING status message. When the callee accepts the call, the callee sends a 200 OK message which is responded to by an ACK message from the caller. This establishes the connection by means of a three way handshake. The users can terminate the connection by means of a BYE message which is followed by an ACK. A brief representation of the SIP architecture and the different avenues of attack is given in Figure 5. There are multiple avenues for flooding in such a scenario. Flooding can be carried out by INVITE messages, BYE messages or REGISTER messages. Also, the attack can be launched against the callee or against the SIP infrastructure resources like the proxy or registrar servers [23]. The major issue here is that after every INVITE or REGISTER message, the connection is maintained by the server for a few minutes. This is to allow users to respond with an authentication or proceed to the next stage of connection establishment. If the attacker sends a large number of such messages, it can effectively tie up the callee or the SIP servers from accepting new connections.

b) Asymmetric DDoS attacks: All the symmetric attacks against the victims also consume a significant amount of resources for the attacker. The objective of the attacker is always to maximize the damage to the victim while minimizing the resources that have to be employed for the attack. This is where asymmetric attacks come into play. Symmetric attacks can be launched and executed without much thought, but asymmetric attacks need careful planning behind them. The idea, however, is the same - to send legitimate requests so that it ties up the resources at the target server. The difference lies in the type of requests used while executing the attack. If one special request can make the Web server perform more work than usual, it is intuitive to try and send this request repeatedly. This will bring down the server faster, using fewer requests and at the same time reduces traffic volume which in turn evades detection.

Assume we have a Web server that connects to a database server. The Web server gives the users to search for jobs based on some search queries. To make the search better, the server employs pattern matching so that the database returns all jobs which have the words the user wants in any scenario. The internal query would look something like:

```
SELECT TOP 10 JobPK, JobDescription
FROM JOBS as j
WHERE JobDescription LIKE '%ASP%'
ORDER BY JobPK
```

This query will typically return the result in under a second. Note that the search term the user enters is “ASP”. However, if the user enters a more complex search term, the query gets complicated. Consider a slight variation of this query as given below

```
SELECT TOP 10 JobPK, JobDescription
FROM JOBS as j
WHERE JobDescription LIKE
'%_[^|?$%"*[(Z*m1_=]-%RT$)|[{34}\?_]||
%TY-3(*.␣>?_!]_%'
ORDER BY JobPK
```

This query performs a complex comparison using regular expressions on the tuples and is likely to tie up the database for some time, possibly minutes.

We refer to the class of requests that do not put excessive load on the server as low workload requests, and the requests which do have to perform significant computation as high workload requests (In reality though, there is no such distinction, and classifying requests based on the workload is purely situational). Assuming that a high workload request performs twice the computation that a low workload request does, it would take just half the amount of requests to exhaust a server. And often times in a real server, the difference isn’t a factor of two, it can be much bigger. The attackers greatly benefit from sending high workload requests to the Web server because it can bring the server down with fewer resources. This attack is termed as an asymmetric workload attack. However, sending high workload requests continuously may raise the suspicion for the Web administrator. This can lead to the connection being closed or blacklisted. A workaround for this is to not use a single connection to deliver multiple high workload requests. Instead, the high workload requests are distributed across multiple connections, each one delivering a small number of requests. In the worst case, each connection may deliver one single high workload request. This attack is called a Repeated One Shot attack [48], and has a two-fold effect on the server. On one hand it makes the Web server to perform more computations and expend more resources. On the other hand, it ties up socket connections which cannot be released until the server has processed the requests.

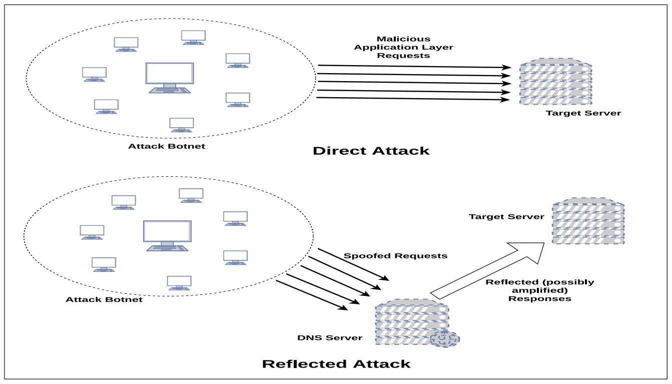


Fig. 6. Direct and Reflected Attacks.

Asymmetric attacks can also be executed in the same way using SOAP requests as well. The trick is to choose SOAP messages that will force the server to perform computation intensive tasks. There is another factor that comes into play for SOAP messages and that is memory. SOAP or XML messages need to be buffered in memory before they can be processed. Unusually large SOAP or XML messages can exhaust memory quickly. To avoid this, many servers will have a size restriction imposed on the XML packets that it receives. To overcome this restriction, attackers can send packets that just satisfy the packet size restriction over multiple connections. Since all of the packets need to be buffered it has the same effect as sending a single large packet but this technique flies under the radar of the defenses imposed by the Web application.

Asymmetric attacks on DNS servers are relatively rare but not unheard of. These attacks focus on open recursive DNS servers. DNS servers can function as recursive or iterative servers. The two types of servers are identical except when they encounter a query to which they do not possess an IP address resolution. An iterative DNS server will find out the address of the DNS server which might potentially have the resolution and send this address over to the client. It is the client’s responsibility to query the server and obtain the IP address. A recursive server on the other hand takes the burden of resolution onto itself and queries the next DNS server till it obtains a resolution. It responds with the final IP address resolution. An attacker might use this information to their advantage and query the server with domain names that are hard to resolve. In the worst case, the attacker can query the server with non-existent domain names so the server will work for a considerable amount of time with no results.

2) Reflected Attacks: For protocols working with an underlying UDP connection, address spoofing is a viable option for attackers and opens up the possibility of a reflected attack (Figure 6). In such an attack, a malicious user spoofs the IP of the target server. Then it proceeds to request the service of either a DNS server or a SIP server pretending to be the victim. The DNS or SIP servers process the request and send the response to the victim directly. If the attacker sends a large number of requests to the DNS or SIP server, the victim server is flooded with a stream of responses which can render it unable to process normal client requests. This mode of attack offers two advantages for attackers. One, the attackers do not need to send direct requests to the server which means that they cannot be identified easily. This mode of attack hides the attackers from detection mechanisms. Second, a small, but well chosen DNS request can generate a large response. This means the attacker needs to expend less resources to flood the target server. The increase in response size compared to the request is different for different protocols and is called the amplification factor. Hence, reflected attacks are also called amplification attacks. DNS amplification attacks are the most prominent attacks in this category and present an extremely difficult challenge. Reflected SIP attacks, even though technically possible, have not been reported in literature widely.

IV. Defending Against DDoS Attacks at the Application Layer

The increased sophistication that goes into executing an application layer DDoS attack makes it comparatively difficult to defend against these attacks. In general, there are two approaches to defend against these attacks - blocking automated requests using user puzzles, or to employ a detection mechanism to identify and block malicious users.

A. Blocking DDoS Attacks Using User Puzzles

The primary cause of a denial of service attack is that attackers are able to send automated requests to the Web server. This is accomplished through the use of botnets or using simple DDoS tools or scripts freely available on the Internet. So the most basic line of defense against any sort of DDoS attack is to restrict automated requests from entering the system. A simple way to achieve this through the use of user puzzles. A user puzzle is any challenge that can be completed easily by a human user, but considerably difficult to complete for an automated system. Simple examples of user puzzles are CAPTCHAs(Completely Automated Public Turing test to tell Computers and Humans Apart) and AYAHs(Are You A Human). Although these puzzles can be broken by bots through the use of suitable image processing algorithms [49]–[51], this simple line of defense against DDoS attacks still works very well. This is because a majority of the DDoS attacks are simple attacks executed using available tools and do not possess the computing power required to crack these challenges.

Associated with this defensive tactic is the question of which users should be served with the challenge-response mechanism. The easiest solution is to deploy the challenge-response mechanism for all the users who want to access your system. This is, however, a relatively inefficient solution. User puzzles significantly reduce the user experience when visiting a website. Users do not want to be burdened by having to solve a puzzle every time they visit a website. The next step would be to deliver the puzzles to only a fraction of the users. SENTRY [52] uses a moderator module which supplies a challenge to a fraction of the requests it encounters. There are provisions within the defense mechanism to sample requests based on the associated workload. Additionally, the complexity of the challenge is also proportional to the workload of the request. However, the users who will be supplied with the puzzle are completely random. Another line of thought is that it is better to serve a challenge only to users who appear to be suspicious. This raises the additional question of how to determine suspicious users. Sivabalan and Radcliffe [53] proposed a way of detecting suspicious users based on the pages viewed and their viewing times. As long as the server load remained within limits, no user is served with a puzzle in their solution. When the server load crosses a threshold, they served suspicious users with AYAHs. This presents a calibration mechanism for their signature generation because if a user with a suspicious signature successfully solves the AYAH, the user is vindicated along with the other users who have a similar signature. On the other hand, repeated failures to solve the AYAH results in blocking of the users.

Similar defense mechanisms are used in Web services as well. Suriadi *et al.* [54] used a hash-based computation-bound puzzle, in which a client is required to find a partial preimage in a cryptographic hash function. They proposed a mechanism to include the puzzle and solution inside the SOAP header. They also used a nonce mechanism to ensure the client does not replay the solution from some other sources. Karnwal *et al.* [55] used a similar idea to detect HTTP DDoS attacks against Web services. Their work also focuses on other attacks like coercive parsing.

While puzzle based mechanisms like CAPTCHAs and AYAHs do work in preventing denial of service attacks, they also reduce the user experience. Hence, most of the research focus is on other approaches to solve the problem.

B. Detecting Application Layer DDoS Attacks

Detecting application layer DDoS attacks presents a significant challenge because of their low traffic volume and the use of legitimate requests. Except in the case of SOAP based attacks, it is almost impossible to detect an attack simply by examining an individual request. Rather, the complex interrelationships among a number of requests need to be modeled and studied to successfully identify an attack at the application layer. In general, attack detection can proceed in four different ways:

1) Template matching of individual requests
2) By tracking requests
3) By analyzing the request stream dynamics
4) By analyzing the request stream semantics.

1) Template Matching: SOAP based DDoS attacks often aim at misusing the flexibility provided by the different security specifications. A common defense mechanism against these attacks is to specify stringent requirements that the incoming requests should obey, such as proper level of nesting or use of external entities. This is called schema hardening [26]. Schema hardening has to be followed by proper verification of the schema requirements in the incoming packets. In other words, by matching each incoming request against a template, SOAP or XML based DDoS attacks can be averted to a large extent.

2) Request Tracking: Request tracking goes one step ahead of inspecting individual requests and matching them to a set template. Request tracking refers to schemes which monitor how many requests (or responses) have been received (or sent out) and possibly identifying correlations between them. Such mechanisms have been employed considerably in the detection of slow DDoS attacks, and attacks that rely on an underlying UDP connection.

3) Analyzing Request Stream Dynamics: Request dynamics analyzes a request stream "by the numbers". Such an analysis is concerned more about statistics rather than about finding meaning in the stream of requests. This refers to the features like number and type of requests, request rate, source IP distribution etc. This layer of observation focuses on the low level details of a request stream and does not focus on how a user views and accesses a Web application. These mechanisms find extensive use in detecting HTTP flooding attacks.

4) Analyzing Request Stream Sematics: These detection mechanisms try to encapsulate features which represent how a user accesses the Web application. A normal user does not consciously know about or manipulate his request rate or his session rate. These users are only concerned about the different resources or Web pages that they need to access and the order in which they need to access them. This layer, therefore, analyzes the different pages the users have requested and the sequence in which users request Web pages. In other words, these detection mechanisms attempt to find an underlying meaning behind the incoming request stream. Detection mechanisms that focus on semantics are used extensively in detecting HTTP flooding attacks, as well as HTTP asymmetric attacks.

V. Defending Against DDoS Attacks Exploiting System Vulnerabilities

Denial of service attacks can be launched exploiting a multitude of vulnerabilities in Web applications. A large number of these vulnerabilities are caused by programmer negligence and lead to opportunities for attacks like SQL or XML injection. SQL injection attempts to inject malicious queries into the database system of the Web application. In the context of DDoS attacks, the attacks aim to put the database in an erratic state which prevents the system from executing. There are a large number of mechanisms available to defend against SQL injection, and considerable amount of research work has gone into this area as well [56]–[60]. Closely related to SQL injection is the vulnerability of XML injection. The idea and execution of the two attacks are similar - the target changes from SQL databases to XML documents or databases. Again, the detection mechanisms are similar to that of SQL injection and a significant amount of work has already gone into the area [61], [62].

Vulnerable algorithms pose another threat to Web applications and attacks like HashDoS and coercive parsing attacks which exploit weaknesses in hashing and parsing algorithms provide ample proof for that. However, despite being a legitimate threat, these attacks can be mitigated quite efficiently by taking necessary precautions. For example, the HashDoS attack was ineffective on hash tables implemented using Perl because Perl used a randomized hashing algorithm which considerably reduced the chance of collisions. Similarly coercive parsing attacks can be eliminated completely by avoiding the use of a DOM based parsing algorithm.

A Web application is made up of a large number of components like the Web server, load balancer, or proxies. Any of these components could have vulnerabilities that lead to a denial of service attack. An earlier version of the Apache server was vulnerable to a Range Header attack. When presented with a large overlapping range of range values in the header, the system CPU utilization peaked and even led to the system crashing. However, system vulnerabilities are considerably rare because of an open and active development community and constant update of software components. The recent versions of Apache are not vulnerable to this attack as a result of proper patches being deployed.

Attacks which exploit system vulnerabilities have already received considerable attention while addressing areas in computer security other than DDoS attacks. The inclusion of this class of attacks in the taxonomy of application layer DDoS attacks is for the sake of completeness because these vulnerabilities, though not strictly under the purview of DDoS attacks, can still be used to launched DDoS attacks. Instead of delving deep into these vulnerabilities, we choose to focus more on the other two classes of vulnerabilities - those exploiting protocol features and system features - because they have received considerably less attention and they form the bulk of application layer DDoS attacks.

VI. Defending Against HTTP Protocol Vulnerabilities

Slow DDoS attacks are the major class of attacks that exploit the HTTP protocol. Cambiaso *et al.* [35] created a detailed taxonomy of slow DDoS attacks. Their classification however, clubs asymmetric attacks and attacks like HashDoS attacks into the category of slow DDoS. The rationale behind that classification was that the request rate in these attacks are much less than that in normal flooding.

A. Preventing Slow DDoS Attacks

There are comparatively fewer research works done on detecting and defending against slow DDoS attacks. However, these attacks can be mitigated by following some preventive mechanisms such as lowering the timeout value of the server, installing suitable Apache security modules, setting up proper IPTables or IDS rules [68]. Park *et al.* [69] conducted simulations that validate the effectiveness of slow DDoS attacks. They observed that the timeout feature in the Web application is effective in reducing the magnitude of the attack but cannot stop the attack from happening.

B. Detecting Slow DDoS Attacks

Slow DDoS attacks are usually detected by employing a request tracking mechanism to keep track of the incomplete

TABLE III
DETECTION MECHANISMS FOR SLOW DDOS ATTACKS

Type of Attack Addressed	Research Work	Features Used	Detection Mechanism	Strengths	Limitations
Slow DDoS attacks	Shtern et al. [63]	Workload, CPU, utilization, CPU time, disk utilization, disk time, waiting time, and throughput	Comparing with baseline metrics	Provides an illusion that the attack is working for attackers	Only works on an SDI
Slow DDoS attacks	Mongelli et al. [64]	Request signal, mutual information	Fourier Analysis	Detects even slow read attacks	Time complexity per observation horizon tends to be more than the duration of the horizon
Slowloris and Slow POST	Tripathi et al. [65]	Probability of complete and incomplete GET and POST requests	Hellinger distance of probability distributions	Low complexity	Able to detect only slow GET and POST attacks
Slowloris, HTTP PRAGMA and HTTP POST attacks	Dantas et al. [39]	Number of packets received per request for each connection	Adaptive Selective Verification	Low complexity	The mechanism chooses a connection to drop at random, so there is a chance a legitimate connection could be dropped
HTTP flooding and Slow DDoS	Katkar et al. [66]	Duration, server error rate, request rate etc.	Naive Bayes	Low complexity	Does not work for complex cases
HTTP flooding and Slow DDoS	Oshima et al. [67]	Request rate	Short term Entropy	Low complexity	Based solely on request rate

requests in the system at any point in time. Some works have also used an analysis of request rate (request dynamics) to identify malicious attackers.

1) Detection Mechanisms Which Employ Request Tracking: Tripathi *et al.* [65] proposed that each connection be associated with a vector denoting the percentage of complete or incomplete GET and POST requests. At any instant of time, if the percentages go beyond a learned threshold, the connection is detected as suspicious. They proposed the use of Hellinger distance to perform the distance calculation. Dantas *et al.* [39] proposed maintaining a record of the number of bytes received for each request in each connection on the server. In this case, if the server runs out of connections, it can randomly choose to either drop the incoming connection or to drop an existing connection taking into consideration the number of received and sent bytes. Their approach defends against slow GET, POST and PRAGMA attacks.

2) Detection Mechanisms Which Analyze the Request Rate: Shtern *et al.* [63] proposed a defense mechanism against slow DDoS attacks based on Software Defined Infrastructure(SDI). SDI is a setup where the network infrastructure is virtualized and the connections and routing tables can be modified on the fly using software controls. This provides a great deal of flexibility. Their approach was to direct suspicious connections to a "shark tank" where they would be analyzed further. However, their approach works only if the infrastructure is software defined and cannot be used in other cases. The work of Giralte *et al.* [70] is one of the few works that seem to be able to detect both HTTP attacks (symmetric and asymmetric) along with Slowloris attacks. Mongelli *et al.* [64] performed Fourier transform analysis on the incoming packet stream to identify slow DDoS attacks. Katkar *et al.* [66] was able to identify slow read and post attacks using a Naive Bayes classifier. Oshima *et al.* [67] proposed a method of using request entropy to identify slow DDoS attacks.

Multiple verb HTTP attacks are not featured prominently in literature because the attack is not well documented. However, it seems plausible that deep packet inspection can evade these attacks.

Slow DDoS attacks are stealthy DDoS attacks that can take down a server with minimal resources. However, defending against these attacks poses a familiar double edged sword. Reducing timeout values and limiting the number of connections continue being the basic line of defense against slow DDoS attacks, but these measures can lead to legitimate users with low bandwidths being forced to relinquishing their connections prematurely. Learning general user behavior continues to be the best defensive option but even with this option, false positives continue to come up. A summary of the detection mechanisms employed against slow DDoS attacks is featured in Table III.

VII. DEFENDING AGAINST XML BASED ATTACKS

DDoS attacks against Web services pose a serious threat to the financial sector because a number of payment gateways operate using Web services. SOAP and JSON are two of the data interchange formats used in Web services, with SOAP being the most common and JSON slowly emerging as the standard. Hence, a lot of the research work on Web services has focused on SOAP. However, majority of these research works focus on how these attacks can be prevented.

A. Preventing XML Denial of Service Attacks

Jensen *et al.* [26] presented a detailed description of all possible attacks on Web services, not just denial of service attacks.

They also propose a number of countermeasures for preventing these attacks. Most important and the most basic among these is schema hardening and proper schema validation. Every Web service has a particular schema which describes the structure of an incoming message. This is usually described in the WSDL (Web Service Description Language). A lot of the time, validation of the incoming message is not performed against this schema due to performance reasons. However, performing such a validation can reduce the incidence of attacks to a large extent. The second part of the solution is schema hardening. Schema hardening involves imposing restrictions (on input size, level of nesting etc.) on the input fields. Additionally this could include removing private functions from the WSDL.

Attacks which rely on oversized security headers or cryptographic functions rely on the fact that there is virtually no limit to the number of levels of nesting that can be performed on the encryption keys or the size of the security header. This can be overcome by including a strict WS-SecurityPolicy which restricts the size of the security header and the nesting of encryption.

The major issue with XML and SOAP messages is that even if the system preprocesses them before execution, they still need to be parsed and stored beforehand. The use of a tree like representation used in DOM parsers necessitates that the entire XML message be present in memory for processing. This is basically the root of all attacks in XML. A solution for this is to use an event driven processing model using an SAX parser. This enables the system to identify an invalid message and abort processing immediately.

B. Detecting XML Denial of Service Attacks

Vissers *et al.* [71] proposed a defense mechanism which incorporates a lot of the suggestions given by Jensen. They developed a mechanism to defend against HTTP and XML denial of service attacks. These two attacks can go hand-in-hand in the case of Web services because HTTP is often used as the delivery mechanism for SOAP or XML messages. Their approach learns a normal user profile, represented by Gaussian models, of several features, such as content length, number of elements, nesting depth, longest element, attribute and namespace. There is a two stage filtering process in this system. In the first stage, the HTTP filter is examined and any abnormalities in content length are ruled out. If the message passes the first stage of processing, the second stage actually parses the messages and cross-verifies the information with that present in the HTTP header. Parsing is carried out using an SAX parser which eliminates the vulnerabilities related to memory and CPU. Padmanabhuni *et al.* [73] proposed a solution involving header and body inspection to detect XML DoS attacks, but suggested using a more compact representation of the schema for faster processing. They selected the Patricia Trie representation for this purpose.

Ficco and Rak [74] analyzed the level of nested XML tags in normal SOAP messages, and created a statistical distribution of the same. If the incoming messages do not abide by the distribution, they are classified as suspicious. If these messages are accompanied by an anomalous CPU usage profile as well, the request is blocked and the system is restored to its usual state.

Chonka *et al.* [72] used a backpropogation neural network on XML features to identify XML DDoS attacks. They also introduced a CTB (Cloud Trace Back) mechanism which marks the requests reaching the cloud server. This is carried out using a reverse proxy server. This marking enables the cloud server to trace back the connections and block them in case of an attack.

XML and SOAP form the backbone of Web services which power financial services and payment gateways. A DDoS attack targeting these systems could cause a large part of the economy to freeze. XML flooding attacks are not represented in literature but the techniques used to detect HTTP flooding could very well be extended to XML as well. What is of considerable importance is how XML and SOAP features can be exploited to launch DDoS attacks. Apart from packet inspection, there are not many defense mechanisms available for these attacks. To a large extent these attacks can be prevented by employing proper schema validation and schema checking. Table IV gives a summary of detection mechanisms for XML DDoS attacks.

VIII. Defending Against HTTP Flooding Attacks

HTTP flooding attacks are similar to network layer flooding attacks in the sense that they rely on a large stream of requests to exhaust the server resources. However the most obvious difference is that they rely on HTTP requests instead of network packets. Even though application layer floods proceed through the use of legitimate user requests, the overall nature of a malicious request stream still differs from a legitimate flow. Detecting application layer floods essentially aims to identify these differences. This is usually done by analyzing request dynamics, request semantics or both. The various research works in the area differ according to the statistical features used, and the machine learning technique employed.

A. Detection Mechanisms Based on Request Dynamics

Since a spike in request rate is one of the biggest telltale signs of a DDoS attack, one line of research focuses on predicting the expected request rate at each instant and scrutinizing the incoming request stream using this knowledge. However, request rate alone is often not a good indicator of an attack, because sophisticated attacks can maintain the illusion of a normal request rate while launching attacks. In such cases, statistics like request entropy can be used.

1) Traffic Estimation: Wen *et al.* [75] uses traffic estimation as the basis for detecting attacks. Their system estimates the expected traffic from historical data using a Kalman filter. If a significant deviation is witnessed from the expected traffic value, this indicates either an attack or a flash crowd. The source IP distribution provides additional evidence to determine if the flood is actually an attack or not. Ni *et al.* [76] used an Adaptive Auto Regressive (AAR) Model to model network traffic. The estimated value is smoothed with a Kalman filter. They introduced a feature

TABLE IV
DETECTION MECHANISMS FOR XML DDOS ATTACKS

Research Work	Features Used	Detection Mechanism	Strengths	Limitations
Vissers et al. [71]	Gaussian models, defined by means and standard deviations of several features, such as content-length, number of elements, nesting depth, longest element, attribute and namespace	Statistical thresholds	Deals with a large number of XML based attacks	Was not tested on a represented dataset
Chonka et al. [72]	XML features like size, rate, nesting	Neural network, along with a traceback mechanism to identify the source	Doesn't just identify the attack, but can reconstruct the path to the attacker as well	Developed to work for both HTTP and XML DoS but shows large variability when operated using a common set of features
Padmanabhuni et al. [73]	Inspection of header and body contents	Patricia Trie representation	Detects a wide variety of XML DoS attacks	Requires packet inspection which is expensive
Ficco & Rak [74]	Level of nesting in SOAP messages	Statistical distributions	Efficient and fast	Works only for deeply nested DoS attacks

called HRPI (HTTP Requests Per IP) as the classification variable. The classification is performed using a Support Vector Machine (SVM).

2) Request Statistics: There are a large number of statistical features that have been used in detecting HTTP floods apart from just the request rate. Some of these features include the request timestamp, IP address, header fields, user agents, number of 200 OK responses, number of error responses and so on. Most research works use a combination of these features for detecting attacks. Yadav and Selvakumar [77] used Principal Component Analysis (PCA) and logistic regression on statistical features to identify attacks. In another work, [78], the same authors applied deep learning on these same statistical features. The authors employed a stacked auto-encoder to extract latent features from the basic statistical features and used logistic regression on these features. Singh *et al.* [79] and Singh and De [80] used a multilayer perceptron (MLP) with the weights trained by a genetic algorithm. In [79] the training features were HTTP GET count, entropy and variance, while in [80] the features used were number of HTTP count, number of the IP addresses, constant mapping function and fixed frame length. Chwalinski *et al.* [81], [82] used the number of requests per resource for each user as the deciding feature and performed K-means clustering to group users into different clusters. They applied likelihood analysis and Bayes factors to determine if an incoming connection can be attributed to an existing cluster or not. A connection that cannot be attributed to any cluster is deemed as malicious. Oo *et al.* [83] extracted features like number of packets, number of bytes, average packet size, packet rate, byte rate, time-interval variance and packet-size variance from connections. If all of these features fall within limits then the connection is most likely benign and is accepted. If all of these values fall outside acceptable limits, the connection is most probably malicious and is blocked. In other cases, where some of these features fall within the specified range and others do not, they employed an HsMM (Hidden semi Markov Model) to efficiently model and classify the connection. Lee *et al.* [84] used Principal Component Analysis (PCA) to reduce the feature dimension and then performed clustering.

Entropy is a much used measure to determine whether a stream of requests is malicious or not. A normal user sends requests which are varying in size, speed and intent. A stream of attack requests on the other hand will be more uniform and have similar requests repeated at regular intervals. As a result, an attack stream will have lower entropy than normal user requests. Devi and Yogesh [85] used this information along with trust values of users to detect attacks. Zhou *et al.* [86] also used model entropy but their work was meant for identifying attack streams in backbone Web traffic. Zhao *et al.* [87] used two measures - Entropy of URL per IP (EUPI) and Entropy of IP per URL (EIPU) for identifying flooding attacks. During a random flooding attack or a fixed URL flooding attack, EUPI would increase, thus indicating an attack. EIPU helped in distinguishing attacks and flash crowds.

B. Detection Mechanisms Based on Request Semantics

Request semantics can be further refined to consist of two classes. Mechanisms that rely on the composition of requests in a request stream, without any consideration to the sequence are much easier to use because of the reduction in data that needs to be analyzed. Mechanisms that rely on request sequence on the other hand tend to be more complex but on the whole tend to be more accurate because they can model normal human behaviour much more accurately.

1) Request Composition: Devi and Yogesh [89] constructed an access matrix using features like HTTP request rate, HTTP session rate, server documents that are accessed and duration of user's access. Singular Value Decomposition (SVD) is applied followed by Independent Component Analysis(ICA) for dimensionality reduction. The incoming connection is assigned a suspicion score based on how much deviation it exhibits from the output of the ICA module. Instead of blocking the connection if it does not match the learned model, this approach schedules the request accordingly. A request which matches with the learned model will have a low suspicion score and consequently will be scheduled for execution faster. A request with a high suspicion score will be scheduled much later. Beitollahi and Deconinck [90] follows a similar approach. The system constructs the CDF (Cumulative Distribution Function) for each of the observed features in the normal user sessions. For an incoming connection, it assigns a suspicion score for each feature based on how likely it is to have the observed value according to the CDF. The final suspicion score is the sum of the individual suspicion scores for all the features.

Beckett *et al.* [93] presented a novel approach to detect DDoS attacks targetting database systems by observing features like the number of databases opened or closed, total and average query time etc. They used a decision tree classifier to identify malicious users based on these features.

2) Request Sequence: Ye *et al.* [88] uses average transition probability and page popularity as features for clustering. They used Euclidean distance and Ward's linkage to match an incoming connection into a cluster. Ndibwile *et al.* [91] proposed the use of three servers - a bait server, a decoy server and a real server - for the detection of attacks. They proposed that all traffic be directed to the bait server initially. From there, proven benign traffic is directed to the real server while suspicious traffic is routed to the decoy server. At the decoy server, traffic is authenticated using decision trees trained using known attack generation tools. Oikonomou and Mirkovic [92] used dynamics, semantics and decoys to weed out attackers. Dynamics refers to features like number of sessions, average pause between sessions, average number of requests per session, and average request inter arrival rate per session. Decision trees were used for classifying the incoming connections. Semantics is modeled by creating a probability graph of the website. The probability of a path is defined as the average of the probabilities of all the edges in the path. Finally, they used decoys which are invisible links or images embedded in the Web page. These links are invisible to normal users and hence do not show up in the traces for normal users. However, the links are still parsed by bots and can be used to identify them.

Table V gives an overview of the different research works, the features and detection mechanisms used, along with advantages and disadvantages where applicable.

HTTP flooding attacks are the most common application layer DDoS attacks. It is often possible to identify a flooding attack by observing the characteristics of the request stream. This is the reason most of the application layer firewalls can provide a great level of defense against HTTP floods. However, systems are far from immune to flooding attacks. Even at the application layer, attack volume is on the rise for flooding attacks. Though still many orders of magnitude less than the network flooding volume, the high volume HTTP floods can clog the firewalls, thus creating a new bottleneck.

IX. Defending Against Asymmetric HTTP Attacks

Compared to HTTP flooding attacks, asymmetric attacks are much harder to detect. This is because of the fact that HTTP floods are marked by a sharp change in many features which can lead to its identification. Asymmetric attacks on the other hand can be executed without raising too many red flags and hence evades detection to a large extent. However, there are certain warning signs that can signal an asymmetric attack in process. Every Web application has a set of normal users who browse the Web application in a certain way. This means that regular users often follow certain paths in the website more often than others. In other words, certain request sequences are more likely than others. Apart from that, normal users take time to browse the Web application. Between every pair of requests issued, there is an associated time gap, often called "think time" because this is when the user processes the Web response and decides what to do next. These two features are often not followed in the case of a request sequence submitted by a bot. This forms the basis of identifying an asymmetric attack.

In other words, request dynamics are unlikely to be of any use in detecting asymmetric attacks. All existing detection mechanisms use request semantics in some form. A summary of the same can be found in Table VI.

A. Detection Mechanisms Based on Request Composition

Ranjan *et al.* [48] was one of the earliest works to take into account the workload profile of a user into account for detecting DDoS attacks. The idea was that normal users are not likely to send high workload requests continuously to the server. Their request profiles would contain interleaved low, medium and high workload requests, while the request sequence of an attacker would have a stream of high and medium workload requests. They analyzed session and request inter arrival distribution, along with client workload profile to assign suspicion scores to users. Based on the deviation of each feature from the legitimate user profile, they assigned suspicion scores to individual users. The connections were scheduled according to the suspicion scores.

B. Detection Mechanisms Based on Request Sequence

Most of the existing works use attack-free user traces to learn how a user accesses the website. Based on this learned model, they calculate the normality of the observed request sequence, which denotes the probability that the incoming user sequence was generated by a legitimate user.

Xu *et al.* [94] modeled the user behavior in a Web application as a probabilistic graph. For each incoming connection, they observed the stream of requests and predict the future request sequence. The similarity between the predicted and observed stream of future requests is used to identify malicious users. Giralte *et al.* [70] presented a three stage detection mechanism which utilizes statistical features, request sequence and request sequence similarity for detecting attacks.

The defense and offense involved in defending against DDoS attacks is strikingly similar to a game between the attacker and the Web administrator. Emami-Taba *et al.* [99] used Game Theory to develop a set of payoff tables to model the attack scenario and used the min-max algorithm to identify attackers.

TABLE V
DETECTION APPROACHES FOR HTTP FLOODING ATTACKS

Research Work	Features Used	Detection Approach	Strengths	Limitations
Wen et al. [75]	Request rate, dispersion of IP addresses	Traffic Estimation (Kalman filter followed by mess extent)	Low complexity, works for flash crowds	Uses only traffic rate as a measure
Ni et al. [76]	HTTP requests per source IP per time(HRPI)	Traffic Estimation (Kalman filter followed by SVM)	Works for flash crowds	Works assuming traffic follows a regular pattern
Chwalinski et al. [81]	Number of requests per resource for a user	Clustering followed by Bayes classification	Addresses attack behavior imitating human behavior	Does not work for short attacking sequences
Chwalinksi et al. [82]	Number of requests per resource for a user	Clustering followed by Likelihood analysis	Addresses attack behavior imitating human behavior	Does not work for short attacking sequences
Ye et al. [88]	IP, reply size, reply code, session details, reply size, object popularity, transition probability, request rate	Clustering	Addresses random flooding attack	Only tested for random GET flooding
Lee et al. [84]	Number of user requests, average number of requests, user interest value, proportion of pages requested, intensity of most frequently visited page, followed by PCA	Clustering	Better detection rate than HsMM for request flooding attacks	Only tested for flooding attacks, higher false positive rate
Devi & Yogesh [89]	HTTP request rate, HTTP session rate, server documents that are accessed and duration of user's access	Access matrix, reduced by SVD and ICA, compared with incoming request features	Works for flash crowds	Request sequence is ignored while constructing the access matrix
Devi & Yogesh [85]	Request Rate	Entropy of Request Rate	Low complexity	Not a suitable candidate for detecting sophisticated attacks
Beitollahi & Deconinck [90]	Request and download rate, uptime, downtime, browsing behaviour-pages, access rate, popularity, hyperlink fraction click, depth- source IP distribution, arrival distribution	Assign scores based on deviation from learned value	Low computational complexity	Attackers can change the page popularity by meek attacks in which case they report a false negative of over 6%
Zhou et al. [86]	Average frequency of resources	Statistical models, followed by request IP entropy	Worked for flash crowds	Not a client side system, instead designed for backbone web traffic
Yadav & Selvakumar [77]	17 features	PCA followed by logistic regression	Low computational complexity	Needs normal and attack dataset
Yadav & Selvakumar [78]	17 features, then used a stacked auto-encoder for deep learning of features	Logistic Regression	Offloads the decision of which feature to use from the user	Needs normal and attack dataset
Ndibwile et al. [91]	Java Script to eliminate bots, request rate, interval	Decision trees	Gives attackers an illusion the attack is working	Needs two additional servers
Oikonomou & Mirkovic [92]	Number of sessions, average pause between sessions, average number of requests per session, and average request inter-arrival rate per session, probability graph, decoys	Decision trees for request dynamics, average path probability for semantics	Detects flash crowds	Using average path probability disguises some attackers
Johnson Singh et al. [79]	HTTP GET count, entropy and variance	Perceptron, with Genetic Algorithm to adjust weights	Provides a high detection rate and low false positive rate for flooding attacks	Takes a long time to converge to a reasonable fitness value for the weights
Singh & De [80]	HTTP count, number of the IP addresses, constant mapping function	Perceptron, with Genetic Algorithm to adjust weights	Provides a high detection rate and low false positive rate for flooding attacks	Takes a long time to converge to a reasonable fitness value for the weights
Oo et al. [83]	Number of packets and bytes, packet size, Packet rate, byte rate, Time-interval and Packet-size variance	Statistical range, followed by HsMM	Does not use HsMM for simple detection cases	Not suitable for sophisticated attacks
Zhao et al. [87]	Entropy of URL per IP (EUPI) and Entropy of IP per URL (EIPU)	Statistical models and entropy threshold	Works well in distinguishing flash crowds	Does not work for attacks leveraging attack workload
Beckett et al. [93]	Number of databases opened and closed, number of database queries, and commits, total and average query time,average number of queries per database open	Decision Tree	Novel approach to mitigate DDoS attacks targetting back-end systems	Does not take into account database query workload

Xie and Yu [95] were the first to use an HsMM for the purpose of detecting denial of service attacks at the application layer. Their work constructed the HsMM from system traces and approximated the think time associated with a page as the number of inline requests the page makes. This can be seen as an approximation of page loading time. They constructed a normal distribution of the likelihoods of the observed sequences which is called Original Likelihood Distribution (OLD). The amount of deviation from the OLD is taken to be the abnormality of an observed request sequence made by a user. A similar approach was also taken by Meng *et al.* [100]. Huang *et al.* [98] also modeled page popularity of a website

TABLE VI
DETECTION MECHANISMS FOR ASYMMETRIC HTTP ATTACKS

Research Work	Features Used	Detection Mechanism	Strengths	Limitations
Xie & Yu [95]	Request sequence and probability, objects requested from server	HsMM (number of inline requests is taken as duration)	Allows for online update	High complexity of M-Algorithm, does not consider dwell time of users as duration
Xie & Yu [96]	Page popularity at different times, modeled as a stochastic process, PCA and ICA for dimensionality reduction	HsMM	Allows for online update	High complexity of detection algorithm
Xu et al. [94]	Page request sequence	Random Walk Graph(Predict the next sequence of requests and matches them with the observed requests using Jacobi coefficients)	Low computational complexity	Unable to model complex user behaviour
Wang et al. [97]	Click ratio(page popularity) both for website and individual users, also transition probability matrix for users and the website	Large Deviation Theorem	Outperformed detection techniques that employ transition probabilities	Click ratio masks request sequence, and hence cannot be used to detect sophisticated attacks
Huang et al. [98]	Page popularity at different times, modeled as a stochastic process, clustering to reduce dimension	HsMM	Able to identify attacks occurring along with a flash crowd	Higher complexity due to the use of HsMM
Giralte et al. [70]	Number of GET requests, mean and standard deviation of GET requests, mean and standard deviation of flows per user, mean and standard deviation of POST requests, flows per minute per user, request per minute per user, request timing in the cache	statistics and graphs	Detects multiple categories of attacks, easy to integrate	Does not consider cases where the attackers may cycle randomly among high workload states
Emami-Taba et al. [99]	Request rate, workload	Game Theory(Minmax-Q algorithm)	Automatic update of payoff tables as the attacker changes strategy	Only tested for simple cases of these attacks
Ranjan et al. [48]	Request and session inter-arrival rate, workload profile similarity	Statistical distributions and probability, KL divergence, suspicion score, scheduling	Low computational complexity, comprehensive	Considers the histogram of workload profile, so request sequence is not considered

using an HsMM. They clustered the available data sets into clusters before using the features to construct an HsMM to reduce the dimensions of the data.

C. Detection Mechanisms by Observing a Change in Page Popularity

There is another approach to detect an application layer attack which relies not on observing the request stream directly, but rather observing the effect of the request stream. Every website has a set of "hot" pages, i.e., pages which are visited more often than others. Generalizing this observation, every page in a website has a probability with which a user accesses it, which denotes how popular the page is. An attack, which does not follow normal user patterns, tends to disrupt this page popularity in haphazard ways. This provides an indirect indication of attack.

Wang *et al.* [97] assumes each user accessing the website has an a priori click ratio which denoted the popularity of the Web pages. The idea is that when a Web application is under attack, the click ratio deviates from the a priori one, and this is used as the means of identification. This work uses large deviation theory to identify how probable a deviation from the a priori click ratio is. They also modeled the click ratio by means of an HsMM and employed Large Deviation theory to measure the probability of deviation. Another work by Xie and Yu [96] constructed an HsMM modeling the document popularity of a Web site. However, they observed that algorithms for building and operating HsMM become considerably complex when using high dimensional data. Hence to reduce the dimensionality of the input data they used traditional dimensionality reduction algorithms of PCA and ICA (Independent Component Analysis).

HsMM proves to be extremely efficient in detecting asymmetric as well as flooding attacks, but they do have some drawbacks. The detection principle involves calculating the probability of the observed sequence at each instant of time. This algorithm however, is complex and thus the overall system load will be more. In some cases, simple statistical calculations prove more efficient than using an HsMM.

Asymmetric HTTP DDoS attacks are extremely severe because of their similarity to normal human behaviour.

TABLE VII
DETECTION MECHANISMS FOR DNS DDOS ATTACKS

Research Work	Features Used	Detection Mechanism	Strengths	Limitations
Sun et al. [101]	DNS requests and responses	Correlation between requests and responses maintained using a bloom filter	Deployable at leaf routers	Possibility of false negatives due to network errors and retransmissions
Khan et al. [102]	DNS requests and responses	Chaos Theory	Lightweight, does not need to be trained offline	Accuracy of only 66%
Kambourakis et al. [103]	DNS request and response matching	Request and response matching	Easy to implement	Not very scalable

Defenses relying on request rate and request statistics will fail to detect these attacks, and by extension they make existing Web application firewalls obsolete. Identifying these attacks is a complex procedure which involves learning normal user behaviour and weeding out connections that show unnatural behaviour. However, most of the techniques used for detecting these attacks rely on an HsMM. While being highly effective, an HsMM has a large computation cost associated with it, which makes it a questionable choice for run time detection. It may also happen that the defense system becomes a bottleneck that attackers can exploit.

X. DEFENDING AGAINST DNS ATTACKS

Denial of Service attacks on DNS servers are the most difficult to detect and are the most devastating. The reason they are so difficult to detect is that DNS relies on UDP messages which means that there is no concept of a connection. This essentially eliminates the possibility of pin-pointing malicious users and blocking them. The best that can be done is that the DNS server identify malicious messages and not waste any processing power in serving those requests. A summary of the detection techniques can be found in Table VII.

A. Mitigation Approaches

Some research has gone into suggesting changes in how the DNS servers maintain records which could potentially help in mitigating attacks on DNS servers. Pappas *et al.* [104] suggested that DNS caches be modified to cache the infrastructure records for longer periods of time. This can help mitigate the effect of denial of service attacks on a DNS server because the caching servers can still function in such a situation. Ballani and Francis [105] went one step further to suggest that DNS servers cache even stale data which can be of some use in case of a DNS outage. These solutions, though, do not prevent attacks on DNS servers, and are difficult to successfully implement.

Zhu *et al.* [106] suggested negotiating a connection using TLS (Transport Layer Security) protocol for DNS communication as well. They called this extension T-DNS, which provided an enhanced privacy and security. While this attack does make it harder to execute an attack on a DNS server, it still leaves the door open for connection oriented attacks, similar to HTTP attacks.

B. Detecting Attacks on DNS Servers

The major issue in detecting an attack on a DNS server is determining if the incoming request is legitimate or not. Guo *et al.* [107] suggested incorporating cookies into DNS messages as a way of identifying if the incoming request is spoofed or not. Rastegari *et al.* [108] developed a detection mechanism for attacks on a DNS server using neural networks. Their approach trains neural networks using features like throughput, average packet size and packet loss to identify attacks.

XI. DEFENDING AGAINST SIP ATTACKS

Considerable work has gone into studying and defending against SIP flooding attacks. Some of these approaches suggested subtle changes in the message structure of SIP while others attempt to detect attacks against SIP end users and proxy servers.

A. Preventing SIP DDoS Attacks

Hussain and Nait-Abdesselam [118] suggested including a field called Critical Number(CN) in the REGISTER message. Every user agent must register with the registrar server before they can receive a call and the CN essentially denotes the maximum number of simultaneous calls they can receive. The proxy server will forward a call to a callee only if the callee is capable of handling the call. This approach however works only by protecting the callee. The proxy server or registrar server are still wide open to attacks. Defending the proxy server however comes with issues of its own. Chen [109] proposed a finite state machine to keep track of the SIP state and ensure it doesn't go into an unidentified state. However, this mechanism has to be deployed before every proxy server which is not cost effective. An ideal solution would be deployed at the client side which will be much more cost efficient.

B. Detecting SIP DDoS Attacks

There are multiple strategies that have been adopted to identify attacks on the client side. A basic defense can employ request tracking and correlation of the SIP messages. Other works have also relied on analyzing the request rate (request dynamics) and modelling the SIP connection as a whole (request semantics).

1) Correlating SIP Messages: One consideration is that during an attack free execution, there is always a strict correlation between the SIP INVITE messages, the response messages and the acknowledgments. In an ideal case of course, for each INVITE message there will be exactly one response and one ACK. However, due to network delays and retransmissions this is not always the case. Geneiatakis *et al.* [110] proposed the use of bloom filters to check whether the number of INVITE requests, responses and ACKs were within the specified limits. Here they allowed the number of responses and ACKs to be double the number of INVITE requests to account for network losses. Sengar *et al.* [111] identified that similar correlations exist between TCPs SYN, ACK and FIN messages as well. They used Hellinger distance to identify if the system was under attack. The data learned from the first *n* time slots were used to identify any malicious behavior in the next slot. If the slot proves to be benign, that slot is also incorporated into the learning data set. This is one way to identify and cope with any on the fly variations. Kumar and Thilagam [116] proposed a different way of correlating SIP messages. They calculated three features - Hash Computation Efficiency (HCE), Successful URI Binding Efficiency (SUBE) and Registration Drop Efficiency (RDE) from the SIP REGISTER messages. These values are used to capture the general attitude of the server which in turn indicates an attack.

2) SIP Request Rate: Tang and Cheng [112] proposed a method to detect SIP flooding attacks when the attack volume does not experience a sudden spike, but rather follows a gradual increase. Most of the existing defenses would not work under these circumstances. The authors proposed the use of Multi Resolution Analysis (MRA) and decomposed the wavelet into two signals - detail signal and approximation signal. The idea was that when the system is under attack, the energy of the detail signal would rise sharply even if the attack volume rises slowly. Allawi *et al.* [113] monitored three features in the SIP communication - request rate, percentage of requests served and average response time. The values of these parameters during the testing phase were compared with thresholds identified during the learning phase to identify an attack. Tang *et al.* [114] used sketch data distributions of four features. Each sketch data is essentially a hash table for the SIP attribute values. The training data set was used to generate a sketch distribution which was compared with the sketch distribution generated during testing. It uses Hellinger distance to measure similarity.

3) Modeling the SIP Connection: Golait and Hubballi [115] modeled the SIP system using a Probabilistic Counting Deterministic Timed Automata (PCDTA). If the system follows the automata from the start state to a final state with the path probability greater than a predetermined threshold, the connection is not anomalous. If the system enters a deadlock or enters an error state, the operation is deemed to be anomalous. Flooding attacks can be detected by using a counter associated with a transition and slow attacks can be identified by the timing probability. Semerci *et al.* [117] monitored the SIP message history of users per time interval. They assigned each user a vector denoting the user's behaviour and the server had a state vector to denote the state of the server, which was nothing but the summary of all user activity. Any significant deviation from the state vector of the server represents an attack.

Table VIII presents a summary of the different ways in which SIP flooding attacks have been detected in literature.

SIP powers a major share of the VoIP communication. Unfortunately, SIP also offers a wide variety of points where a denial of service attack can be executed. Detecting these attacks by observing merely the traffic flow volume may result in false positives, and hence sophisticated techniques which model the SIP architecture and detect anomalies are needed. However, the performance and efficacy of these techniques have to be considered to ensure it can operate in real time. Also, SIP provides multiple points of attack, all of which must be secured to ensure that the entire system works perfectly.

XII. Defending Against Reflected Attacks

Reflected attacks present a significant challenge in situations where the underlying transport layer protocol is connectionless. This type of attack allows the attackers the opportunity to remain hidden while simultaneously employ an asymmetric attack on the victim. In cases where the response messages reflected on to the victim is much larger than the request messages sent by the attacker, the attack is called an amplification attack. SIP reflection attacks have not been reported in literature much, and the majority of reflected attacks are accounted for by reflected DNS attacks. As a result, we focus our attention on reflected DNS attacks only.

Almost all works that attempt to defend against reflected DDoS attacks at the application layer attempt to employ some sort of request tracking mechanism. They attempt to identify some correlation between the number of requests and responses and hence try to separate legitimate users from malicious ones. The type of correlation identified, and the mechanism used to perform the correlation are what separate the different works in the area.

Kambourakis *et al.* [103] presented a simple solution to DNS amplification attacks. They proposed that when a DNS amplification attack takes place, the targeted DNS server receives responses without having previously sent out the corresponding request. Their approach simply requires that such data (orphan pairs) be immediately classified as suspicious and discarded. Sun *et al.* [101] used a similar approach by storing the outgoing DNS requests using a bloom filter. If the number of responses coming in is greater than the number of requests, this signifies an attack. Any response which does not match the request information stored is discarded. Khan *et al.* [102] used chaos theory to identify the incoming DNS stream as malicious or not.

DDoS attacks against DNS extremely hard to detect and at the same time extremely devastating as well because it can potentially affect a large number of other sites and can even disrupt Internet temporarily. The use of UDP which is a connectionless protocol essentially makes this a very promising line of attack for attackers.

TABLE VIII
DETECTION MECHANISMS FOR SIP FLOODING ATTACKS

Research Work	Features Used	Detection Mechanism	Strengths	Limitations
Chen [109]	SIP transaction model	FSM, there is an upper bound on how many times a user can go to an invalid state in the FSM	Does not require modification of existing SIP software	Cannot detect malformed and spoofed messages, is not that effective in the case of flooding, but works well for error messages
Geneiatakis et al. [110]	Request rate and type	Matches invite and ACK messages, raises an alarm if there is an anomaly	Negligible detection time	Possibility of false negatives
Sengar et al. [111]	Number of SIP messages of each time in unit time	Hellinger distance	Fast	Fails for low rate attacks
Tang & Cheng [112]	Request rate	Multi Resolution Analysis using DWT	Works for stealthy SIP floods	Depends only on request rate
Allawi et al. [113]	Request rate, percentage of served requests per second, mean value of server request/response delays	Comparing with thresholds	Simple	Unlikely to work for all types of attacks
Tang et al. [114]	Rate and timing of INVITE, OK, ACK, BYE	Hellinger distance between sketch data	Detects attacks spanning multiple attributes	Fails to detect stealthy attacks
Golait & Hubballi [115]	SIP message timings and rate	PCDTA	Detects all kinds of attacks on SIP	Requires a separate model for each connection so memory requirement is linear
Kumar & Thilagam [116]	Hash Computation Efficiency (HCE), Successful URI Binding Efficiency (SUBE) and Registration Drop Efficiency (RDE)	Statistical analysis to determine attitude of the server	Works for low rate attacks	Considers only the REGISTER request
Semerci et al.[117]	SIP message count and sequence	Statistical distance measure (Mahalanobis distance) to identify deviation from normal state	Works on the fly and does not require any previous data for building a model	Susceptible to slowly rising attack volume

XIII. TOOLS AND DATASETS RELATED TO APPLICATION LAYER DDOS ATTACKS

A. *Tools for Defending Against Application Layer DDoS Attacks*

Specially designed application layer firewalls are necessary to detect application layer DDoS attacks, because these attacks can only be detected through inspecting HTTP requests and identifying patterns in the request flow. There are open source and commercial firewalls available that are able to detect and block these attacks, but these tools still focus on the primitive application layer DDoS attacks. A comparison of some of the commercial defense tools are given in Table IX. There are a few open source DDoS defense tools available but they lack in strength and diversity. Apache provides a module mod_evasive which can be configured to block connections with predefined rules. For example, it could be set to block connections which send requests over a particular threshold rate, or attempt to open too many connections. HAProxy is an open source load balancer which can be configured to block malicious users in the same way. It can also detect a TCP SYN flood and Slowloris attacks. Radware has deployed an open source DDoS defense system called Opendaylight, but it depends on an SDI to operate. Google has a DDoS defense mechanism called Project Shield, which allows websites to route their traffic through Google servers, where the traffic will be filtered.

While the existing defense mechanisms do work well for the common application layer DDoS attacks like HTTP floods, they rely on predefined rules and do not consider normal user behaviour for detecting attacks. As a result, asymmetric attacks are likely to fly under the radar of these defenses.

B. *Datasets for Application Layer DDoS*

There is a dearth of datasets available for training and testing application layer DDoS defense mechanisms. While there are a wealth of network traces available, there are comparatively few application layer traces available, and most of the available datasets are old. There are a number of sources of attack free HTTP traces provided by the Internet Traffic Archives. Some of these datasets are:

- WorldCup98 - 1.3 billion Web requests recorded at servers for the 1998 World Cup.
- EPA-HTTP - a day of HTTP logs from a busy WWW server.
- SDSC-HTTP - a day of HTTP logs from a busy WWW server.
- Calgary-HTTP - a year of HTTP logs from a CS departmental WWW server.
- ClarkNet-HTTP - two weeks of HTTP logs from a busy Internet service provider WWW server.
- NASA-HTTP - two months of HTTP logs from a busy WWW server.
- Saskatchewan-HTTP - seven months of HTTP logs from a University WWW server.

However, most of these datasets date back to the late 1990s or early 2000s. Shiravi *et al.* [119] recognized this problem and

TABLE IX
COMPARISON OF COMMERCIAL APPLICATION LAYER DDoS DEFENSE TOOLS

Firewall Designer	**Slow Attacks**	**Flooding**	**DNS Protection**	**XML Attacks**
Imperva	✓	✓	✓	
Corero		✓	✓	
Akamai		✓	✓	
Cloudflare		✓	✓	
Arbor		✓		
Sucuri	✓	✓	✓	
Fortiddos		✓	✓	
F5	✓	✓	✓	✓

generated a dataset that has been used in some of the recent research works featuring HTTP GET flood attacks [120]. There are a few datasets which describe a DDoS attack which are given below.

- *CAIDA:* Contains approximately one hour of anonymized traffic traces from a DDoS attack on August 4, 2007.
- *DARPA:* DARPA contains DDoS datasets of varying complexity of attack, but the dataset is from the year 2000. Apart from that, analyses on the dataset have led some researchers to suggest it does not resemble real network data.

However, these datasets are not very reliable and so most of the research works have relied upon generated attack traces using existing DDoS attack generation tools. Some of these attack tools are listed below.

- *Low Orbit Ion Cannon:* LOIC is a simple tool which can be used to generate TCP, UDP or HTTP floods.
- *HULK:* HULK generates a random flood attack capable of bypassing proxies and evade detection.
- *DDoSim:* DDoSim is a tool which generates an HTTP or TCP flood with either valid or invalid requests.
- *R-U-D-Y:* R-U-D-Y which stands for 'aRe yoU Dead Yet', is used to generate a low POST attack on a Web server.
- *Tor's Hammer:* Tor's Hammer is another attack tool which uses the slow POST attack.
- Pyloris: It is a python script used to test for connection exhaustion DoS vulnerabilities in HTTP, FTP, SMTP, IMAP, and Telnet.
- *Golden Eye:* Golden Eye is a stress testing tool for HTTP protocol.
- *SIPp-DD:* SIPp is a SIP traffic generation tool, which has been leveraged by the work of Stanek and Kenkl [121] to develop SIPp-DD. This tool is capable of generating SIP flooding attacks.

XIV. NECESSARY FEATURES FOR DEFENSE MECHANISMS

DDoS detection mechanisms have to work efficiently and must be capable of detecting and blocking attacks at runtime. The following features are crucial for a DDoS detection mechanism:

- *Scalability:* As the number of users (malicious and legitimate) rises, the load on the detection mechanism also rises. If the detection mechanism is unable to scale up to meet the growing demand, it becomes a bottleneck that the attackers can exploit. Any DDoS defense mechanism must be able to detect and block attackers in the least possible time. Apart from that the mechanism should be able to handle a large number of users and not crumble under the load of the attack.
- *Detection Speed:* The detection mechanism must be capable of evaluating a connection swiftly to avoid delay. A long delay in turn leads to a longer response time which degrades the user experience.
- *Low computation overhead:* The detection mechanism works in parallel with the Web server and must use as little computational resources as possible. This is in line with the above point where the existence of a firewalls should not in any way degrade the quality of service. A high computation overhead also means that the firewall can be used as a bottleneck by the attackers and thus presents a new avenue of attack.
- *Detection Accuracy:* Detection accuracy can be measured using two factors: detection rate and false negatives. Detection rate denotes how much of the attack connections the firewall recognized. False negative rate denotes the opposite, i.e., how much of the incoming attack traffic did the firewall pass on to the Web server.
  Let N_t denote the total number of attack vectors used during testing. Let N_r be the total number of connections reported as malicious, and N_d be the number of connections correctly reported as malicious.

$$Detection\ Rate\ (DR) = \frac{N_d}{N_t} \tag{1}$$

$$False\ Negative\ Rate\ (FNR) = \frac{N_t - N_d}{N_t}. \tag{2}$$

- Must not penalize legitimate users: This is a very important consideration for any organization. A DDoS attack system which labels legitimate users as malicious takes away valuable business from the organization. False Positive Rate is used to calculate the degree to which a detection system classifies legitimate users as malicious.

$$False\ Positive\ Rate\ (FPR) = \frac{N_r - N_d}{N_r}. \tag{3}$$

XV. CONCLUSION

In this work, a detailed description and taxonomy of application layer distributed denial of service attacks has been presented to aid researchers in better understanding and dealing with the dangers that these attacks present. A review of the existing research directions and defense mechanisms has also been presented to bring out the different features used for detecting these attacks, and the different methods of detection. Even though a reasonable amount of work has gone into detecting and defending against application layer denial of service attacks, they still remain a major threat because of the difficulty in adopting the defenses. We hope that this work will open doors for further discussion and research into this area.

References

[1] Radware. (2012). *Cyber Security on the Offense—A Study of IT Security Experts*. [Online]. Available: https://security.radware.com/uploadedfiles/resources_and_content/attack_tools/cybersecurityontheoffense.pdf

[2] Arbor Networks. (2016). *2016 DDoS Attack Statistics*. [Online]. Available: https://www.arbornetworks.com/arbor-networks-releases-global-ddos-attack-data-for-1h-2016

[3] Infosecurity Magazine. (2017). *Anonymous Attacks Spanish Government Sites*. [Online]. Available: https://www.infosecurity-magazine.com/news/anonymous-attacks-spanish/

[4] Incapsula. (2015). *Analysis of Vikingdom DDoS Attacks on U.S. Government Sites*. [Online]. Available: https://www.incapsula.com/blog/vikingdom-ddos-attacks-us-government.html

[5] Silicon. (2017). *Irish Government Websites Taken Down By DDoS Attacks*. [Online]. Available: http://www.silicon.co.uk/e-regulation/irish-government-websites-ddos-184428

[6] Register. (2012). *Anonymous Turns Its DDoS Cannons on India*. [Online]. Available: https://www.theregister.co.uk/2012/05/18/anonymous_ddos_india_sites/

[7] Corero. (2016). *DDoS Attacks Plague Olympic & Brazilian Government Websites*. [Online]. Available: https://www.corero.com/blog/749-ddos-attacks-plague-olympic–brazilian-government-websites.html

[8] Register. (2018). *Gits Club GitHub Code Tub With Record-Breaking 1.35Tbps DDoS Drub*. [Online]. Available: https://www.theregister.co.uk/2018/03/01/github_ddos_biggest_ever/

[9] Coindesk. (2017). *Bitcoin Gold Website Down Following DDoS Attack*. [Online]. Available: https://www.coindesk.com/bitcoin-gold-website-following-massive-ddos-attack/

[10] Guardian. (2016). *HSBC Suffers Online Banking Cyber Attack*. [Online]. Available: https://www.theguardian.com/money/2016/jan/29/hsbc-online-banking-cyber-attack

[11] Arbor Networks. (2012). *Leading U.S. Banks Targeted in DDoS Attacks*. [Online]. Available: https://nakedsecurity.sophos.com/2012/09/27/banks-targeted-ddos-attacks/

[12] Dyn. (2016). *Dyn Analysis Summary Of Friday October 21 Attack*. [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[13] International Times. (2016). *Hackers Leave Finnish Residents Cold After DDoS Attack Knocks Out Heating Systems*. [Online]. Available: http://www.ibtimes.co.uk/hackers-leave-finnish-residents-cold-after-ddos-attack-knocks-out-heating-systems-1590639

[14] SCMagazine. (2017). *DDoS Attacks Delay Trains, Stymie Transportation Services in Sweden*. [Online]. Available: https://www.scmagazine.com/ddos-attacks-delay-trains-stymie-transportation-services-in-sweden/article/700227/

[15] Guardian. (2016). *Massive Cyber-Attack Grinds Liberia's Internet to a Halt*. [Online]. Available: https://www.theguardian.com/technology/2016/nov/03/cyberattack-internet-liberia-ddos-hack-botnet

[16] Forbes. (2014). *Bitcoin Hit By Massive DDoS Attack As Tensions Rise*. [Online]. Available: https://www.forbes.com/sites/leoking/2014/02/12/bitcoin-hit-by-massive-ddos-attack-as-tensions-rise/

[17] Guardian. (2017). *Critical Infrastructure Not Ready for DDoS Attacks: FOI Data Report*. [Online]. Available: https://www.scmagazineuk.com/critical-infrastructure-not-ready-for-ddos-attacks-foi-data-report/article/684838/

[18] Incapsula. (2016). *DDoS Threat Landscape Report 2015–16*. [Online]. Available: https://lp.incapsula.com/rs/804-TEY-921/images/2015-16%20DDoS%20Threat%20Landscape%20Report.pdf

[19] L. Garber, "Denial-of-service attacks rip the Internet," *Computer*, vol. 33, no. 4, pp. 12–17, 2000.

[20] I. Ristic. (2011). *TLS Renegotiation and Denial of Service Attacks*. [Online]. Available: https://blog.qualys.com/ssllabs/2011/10/31/tls-renegotiation-and-denial-of-service-attacks

[21] Incapsula. (2017). *Global DDoS Threat Landscape Q1 2017*. [Online]. Available: https://www.incapsula.com/ddos-report/ddos-report-q1-2017.html

[22] Kaspersky. (2016). *Kaspersky DDoS Intelligence Report for Q1 2016*. [Online]. Available: https://securelist.com/kaspersky-ddos-intelligence-report-for-q1-2016/74550/

[23] D. Geneiatakis *et al.*, "Survey of security vulnerabilities in session initiation protocol," *IEEE Commun. Surveys Tuts.*, vol. 8, no. 3, pp. 68–81, 3rd Quart., 2006.

[24] S. Armoogum and N. Mohamudally, "Survey of practical security frameworks for defending sip based VoIP systems against DoS/DDoS attacks," in *Proc. IEEE IST Africa Conf.*, 2014, pp. 1–11.

[25] I. Hussain, S. Djahel, Z. Zhang, and F. Naït-Abdesselam, "A comprehensive study of flooding attack consequences and countermeasures in session initiation protocol (SIP)," *Security Commun. Netw.*, vol. 8, no. 18, pp. 4436–4451, 2015.

[26] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on Web services," *Comput. Sci. Res. Develop.*, vol. 24, no. 4, pp. 185–197, 2009.

[27] V. Durcekova, L. Schwartz, and N. Shahmehri, "Sophisticated denial of service attacks aimed at application layer," in *Proc. ELEKTRO*, May 2012, pp. 55–60.

[28] M. Aamir and M. A. Zaidi, "A survey on DDoS attack and defense strategies: From traditional schemes to current techniques," *Interdiscipl. Inf. Sci.*, vol. 19, no. 2, pp. 173–200, 2013.

[29] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2046–2069, 4th Quart., 2013.

[30] E. Cambiaso, G. Papaleo, G. Chiola, and M. Aiello, "Slow DoS attacks: Definition and categorisation," *Int. J. Trust Manag. Comput. Commun.*, vol. 1, nos. 3–4, pp. 300–319, 2013.

[31] N. S. Vadlamani, "A survey on detection and defense of application layer DDoS attacks," M.S. thesis, Univ. Nevada, Reno, NV, USA, 2013.

[32] Y. Wang, L. Liu, B. Sun, and Y. Li, "A survey of defense mechanisms against application layer distributed denial of service attacks," in *Proc. 6th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, 2015, pp. 1034–1037.

[33] G. Mantas, N. Stakhanova, H. Gonzalez, H. H. Jazi, and A. A. Ghorbani, "Application-layer denial of service attacks: Taxonomy and survey," *Int. J. Inf. Comput. Security*, vol. 7, nos. 2–4, pp. 216–239, 2015.

[34] K. Singh, P. Singh, and K. Kumar, "Application layer HTTP-get flood DDoS attacks: Research landscape and challenges," *Comput. Security*, vol. 65, pp. 344–372, Mar. 2017.

[35] E. Cambiaso, G. Papaleo, and M. Aiello, "Taxonomy of slow DoS attacks to Web applications," in *Proc. Int. Conf. Security Comput. Netw. Distrib. Syst.*, 2012, pp. 195–204.

[36] Apache. (2011). *Apache HTTPD Security Advisory*. [Online]. Available: https://httpd.apache.org/security/CVE-2011-3192.txt

[37] CCC. (2011). *Effective Denial of Service Attacks Against Web Application Platforms*. [Online]. Available: https://events.ccc.de/congress/2011/Fahrplan/events/4680.en.html

[38] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *Proc. USENIX Security Symp.*, 2003, pp. 29–44.

[39] Y. G. Dantas, V. Nigam, and I. E. Fonseca, "A selective defense for application layer DDoS attacks," in *Proc. IEEE Joint Intell. Security Informat. Conf. (JISIC)*, The Hague, The Netherlands, 2014, pp. 75–82.

[40] DDoS-Guard. (2014). *Single Request HTTP Flood (Multiple VERB Single Request)*. [Online]. Available: https://ddos-guard.net/en/terminology/single-request-http-flood-multiple-verb-single-request

[41] D. Beckett and S. Sezer, "HTTP/2 cannon: Experimental analysis on HTTP/1 and HTTP/2 request flood DDoS attacks," in *Proc. IEEE 7th Int. Conf. Emerg. Security Technol. (EST)*, Canterbury, U.K., 2017, pp. 108–113.

[42] Imperva. (2016). *HTTP/2: In-Depth Analysis of the Top Four Flaws of the Next Generation Web Protocol*. [Online]. Available: https://www.imperva.com/docs/Imperva_HII_HTTP2.pdf

[43] OWASP. (2017). *XML External Entity (XXE) Processing*. [Online]. Available: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

[44] T. D. Morgan and O. A. Ibrahim. (2014). *XML Schema, DTD, and Entity Attacks: A Compendium of Known Techniques*. [Online]. Available: http://www. vsecurity. com/download/papers/XMLDTDEntityAttacks. pdf

[45] Cloudflare. (2014). *HTTP Flood Attack*. [Online]. Available: https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/

[46] WS-Attacks.org. (2010). *XML Flooding*. [Online]. Available: http://www.ws-attacks.org/XML_Flooding

[47] Incapsula. (2010). *DNS Flooding*. [Online]. Available: https://www.incapsula.com/ddos/attack-glossary/dns-flood.html

[48] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly, "DDoS-shield: DDoS-resilient scheduling to counter application layer attacks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 26–39, Feb. 2009.

[49] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1. Madison, WI, USA, 2003, pp. I–I.

[50] J. Yan and A. S. El Ahmad, "Breaking visual CAPTCHAs with naive pattern recognition algorithms," in *Proc. IEEE 23rd Annu. Comput. Security Appl. Conf. (ACSAC)*, Miami Beach, FL, USA, 2007, pp. 279–291.

[51] S. Sivakorn, J. Polakis, and A. D. Keromytis, "I'm not a human: Breaking the Google reCAPTCHA," in *Proc. Black Hat*, 2016, pp. 1–12.

[52] H. Zhang, A. Taha, R. Trapero, J. Luna, and N. Suri, "Sentry: A novel approach for mitigating application layer DDoS threats," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, 2016, pp. 465–472.

[53] S. Sivabalan and P. J. Radcliffe, "A novel framework to detect and block DDoS attack at the application layer," in *Proc. IEEE TENCON Spring Conf.*, Sydney, NSW, Australia, 2013, pp. 578–582.

[54] S. Suriadi, D. Stebila, A. Clark, and H. Liu, "Defending Web services against denial of service attacks using client puzzles," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Washington, DC, USA, 2011, pp. 25–32.

[55] T. Karnwal, T. Sivakumar, and G. Aghila, "A comber approach to protect cloud computing against XML DDoS and HTTP DDoS attack," in *Proc. IEEE Students' Conf. Elect. Electron. Comput. Sci. (SCEECS)*, Bhopal, India, 2012, pp. 1–5.

[56] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and monitoring for NEutralizing SQL-injection attacks," in *Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Long Beach, CA, USA, 2005, pp. 174–183.

[57] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," in *Proc. 5th Int. Workshop Softw. Eng. Middleware*, Lisbon, Portugal, 2005, pp. 106–113.

[58] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL injection attacks," in *Proc. Int. Conf. Appl. Cryptography Netw. Security*, 2004, pp. 292–302.

[59] R. Chandrashekhar, M. Mardithaya, S. Thilagam, and D. Saha, "SQL injection attack mechanisms and prevention techniques," in *Proc. Int. Conf. Adv. Comput. Netw. Security*, Surathkal, India, 2011, pp. 524–533.

[60] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proc. IEEE Int. Symp. Secure Softw. Eng.*, vol. 1, 2006, pp. 13–15.

[61] N. Palsetia, G. Deepa, F. A. Khan, P. S. Thilagam, and A. R. Pais, "Securing native xml database-driven Web applications from XQuery injection vulnerabilities," *J. Syst. Softw.*, vol. 122, pp. 93–109, Dec. 2016.

[62] G. Deepa *et al.*, "Black-box detection of XQuery injection and parameter tampering vulnerabilities in Web applications," *Int. J. Inf. Security*, vol. 17, no. 1, pp. 105–120, 2018.

[63] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou, "Towards mitigation of low and slow application DDoS attacks," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Boston, MA, USA, 2014, pp. 604–609.

[64] M. Mongelli, M. Aiello, E. Cambiaso, and G. Papaleo, "Detection of dos attacks through Fourier transform and mutual information," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 7204–7209.

[65] N. Tripathi, N. Hubballi, and Y. Singh, "How secure are Web servers? An empirical study of slow HTTP DoS attacks and detection," in *Proc. 11th Int. Conf. Availability Rel. Security (ARES)*, Salzburg, Austria, Aug. 2016, pp. 454–463.

[66] V. Katkar, A. Zinjade, S. Dalvi, T. Bafna, and R. Mahajan, "Detection of DoS/DDoS attack against HTTP servers using naive Bayesian," in *Proc. Int. Conf. Comput. Commun. Control Autom.*, Pune, India, Feb. 2015, pp. 280–285.

[67] S. Oshima, T. Nakashima, and T. Sueyoshi, "Early DoS/DDoS detection method using short-term statistics," in *Proc. Int. Conf. Complex Intell. Softw. Intensive Syst. (CISIS)*, Kraków, Poland, 2010, pp. 168–173.

[68] D. Moustis and P. Kotzanikolaou, "Evaluating security controls against HTTP-based DDoS attacks," in *Proc. IISA*, Piraeus, Greece, Jul. 2013, pp. 1–6.

[69] J. Park, K. Iwai, H. Tanaka, and T. Kurokawa, "Analysis of slow read DoS attack," in *Proc. IEEE Int. Symp. Inf. Theory Appl. (ISITA)*, Melbourne, VIC, Australia, 2014, pp. 60–64.

[70] L. C. Giralte, C. Conde, I. M. De Diego, and E. Cabello, "Detecting denial of service by modelling Web-server behaviour," *Comput. Elect. Eng.*, vol. 39, no. 7, pp. 2252–2262, 2013.

[71] T. Vissers, T. S. Somasundaram, L. Pieters, K. Govindarajan, and P. Hellinckx, "DDoS defense system for Web services in a cloud environment," *Future Gener. Comput. Syst.*, vol. 37, pp. 37–45, Jul. 2014.

[72] A. Chonka, Y. Xiang, W. Zhou, and A. Bonti, "Cloud security defence to protect cloud computing against HTTP-DoS and XML-DoS attacks," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1097–1107, 2011.

[73] S. Padmanabhuni, V. Singh, K. M. S. Kumar, and A. Chatterjee, "Preventing service oriented denial of service (PreSODoS): A proposed approach," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Chicago, IL, USA, 2006, pp. 577–584.

[74] M. Ficco and M. Rak, "Intrusion tolerant approach for denial of service attacks to Web services," in *Proc. IEEE 1st Int. Conf. Data Compression Commun. Process. (CCP)*, Palinuro, Italy, 2011, pp. 285–292.

[75] S. Wen, W. Jia, W. Zhou, W. Zhou, and C. Xu, "CALD: Surviving various application-layer DDoS attacks that mimic flash crowd," in *Proc. IEEE 4th Int. Conf. Netw. Syst. Security (NSS)*, Melbourne, VIC, Australia, 2010, pp. 247–254.

[76] T. Ni, X. Gu, H. Wang, and Y. Li, "Real-time detection of application-layer DDoS attack using time series analysis," *J. Control Sci. Eng.*, vol. 2013, p. 4, Aug. 2013.

[77] S. Yadav and S. Selvakumar, "Detection of application layer DDoS attack by modeling user behavior using logistic regression," in *Proc. IEEE 4th Int. Conf. Rel. Infocom Technol. Optim. (ICRITO) (Trends Future Directions)*, Noida, India, 2015, pp. 1–6.

[78] S. Yadav and S. Subramanian, "Detection of application layer DDoS attack by feature learning using stacked autoencoder," in *Proc. IEEE Int. Conf. Comput. Techn. Inf. Commun. Technol. (ICCTICT)*, New Delhi, India, 2016, pp. 361–366.

[79] K. J. Singh, K. Thongam, and T. De, "Entropy-based application layer DDoS attack detection using artificial neural networks," *Entropy*, vol. 18, no. 10, p. 350, 2016.

[80] K. J. Singh and T. De, "MLP-GA based algorithm to detect application layer DDoS attack," *J. Inf. Security Appl.*, vol. 36, pp. 145–153, Oct. 2017.

[81] P. Chwalinski, R. Belavkin, and X. Cheng, "Detection of application layer DDoS attacks with clustering and Bayes factors," in *Proc. IEEE Int. Conf. Syst. Man Cybern. (SMC)*, Manchester, U.K., 2013, pp. 156–161.

[82] P. Chwalinski, R. Belavkin, and X. Cheng, "Detection of application layer DDoS attack with clustering and likelihood analysis," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Atlanta, GA, USA, 2013, pp. 217–222.

[83] K. K. Oo, K. Z. Ye, H. Tun, K. Z. Lin, and E. Portnov, "Enhancement of preventing application layer based on DDoS attacks by using hidden semi-Markov model," in *Genetic and Evolutionary Computing*. Cham, Switzerland: Springer, 2016, pp. 125–135.

[84] S. Lee, G. Kim, and S. Kim, "Sequence-order-independent network profiling for detecting application layer DDoS attacks," *EURASIP J. Wireless Commun. Netw.*, vol. 2011, no. 1, p. 50, 2011.

[85] S. R. Devi and P. Yogesh, "Detection of application layer DDoS attacks using information theory based metrics," in *Proc. CS IT-CSCP*, vol. 10, 2012, pp. 213–223.

[86] W. Zhou, W. Jia, S. Wen, Y. Xiang, and W. Zhou, "Detection and defense of application-layer DDoS attacks in backbone Web traffic," *Future Gener. Comput. Syst.*, vol. 38, pp. 36–46, Sep. 2014.

[87] Y. Zhao, W. Zhang, Y. Feng, and B. Yu, "A classification detection algorithm based on joint entropy vector against application-layer DDoS attack," *Security Commun. Netw.*, vol. 2018, pp. 1–8, Aug. 2018.

[88] C. Ye, K. Zheng, and C. She, "Application layer DDoS detection using clustering analysis," in *Proc. 2nd Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Changchun, China, 2012, pp. 1038–1041.

[89] S. R. Devi and P. Yogesh, "An effective approach to counter application layer DDoS attacks," in *Proc. 3rd Int. Conf. Comput. Communi. Netw. Technol. (ICCCNT)*, Coimbatore, India, 2012, pp. 1–4.

[90] H. Beitollahi and G. Deconinck, "ConnectionScore: A statistical technique to resist application-layer DDoS attacks," *J. Ambient Intell. Humanized Comput.*, vol. 5, no. 3, pp. 425–442, 2014.

[91] J. D. Ndibwile, A. Govardhan, K. Okada, and Y. Kadobayashi, "Web server protection against application layer DDoS attacks using machine learning and traffic authentication," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 3. Taichung, Taiwan, 2015, pp. 261–267.

[92] G. Oikonomou and J. Mirkovic, "Modeling human behavior for defense against flash-crowd attacks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dresden, Germany, 2009, pp. 1–6.

[93] D. Beckett, S. Sezer, and J. McCanny, "New sensing technique for detecting application layer DDoS attacks targeting back-end database resources," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, pp. 1–7.

[94] C. Xu, G. Zhao, G. Xie, and S. Yu, "Detection on application layer DDoS using random walk model," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, 2014, pp. 707–712.

[95] Y. Xie and S.-Z. Yu, "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 54–65, Feb. 2009.

[96] Y. Xie and S.-Z. Yu, "Monitoring the application-layer DDoS attacks for popular Websites," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 15–25, Feb. 2009.

[97] J. Wang, X. Yang, and K. Long, "Web DDoS detection schemes based on measuring user's access behavior with large deviation," in *Proc. IEEE Glob. Telecommun. Conf. (GLOBECOM)*, Kathmandu, Nepal, 2011, pp. 1–5.

[98] C. Huang, J. Wang, G. Wu, and J. Chen, "Mining Web user behaviors to detect application layer DDoS attacks," *J. Softw.*, vol. 9, no. 4, pp. 985–990, 2014.

[99] M. Emami-Taba, M. Amoui, and L. Tahvildari, "Strategy-aware mitigation using Markov games for dynamic application-layer attacks," in *Proc. IEEE 16th Int. Symp. High Assurance Syst. Eng. (HASE)*, 2015, pp. 134–141.

[100] B. Meng, W. Andi, X. Jian, and Z. Fucai, "DDOS attack detection system based on analysis of users' behaviors for application layer," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE) Embedded Ubiquitous Comput. (EUC)*, vol. 1. Guangzhou, China, 2017, pp. 596–599.

[101] C. Sun, B. Liu, and L. Shi, "Efficient and low-cost hardware defense against DNS amplification attacks," in *Proc. IEEE Glob. Telecommun. Conf. (IEEE GLOBECOM)*, New Orleans, LA, USA, 2008, pp. 1–5.

[102] M. S. Khan, K. Ferens, and W. Kinsner, "A chaotic measure for cognitive machine classification of distributed denial of service attacks," in *Proc. IEEE 13th Int. Conf. Cogn. Informat. Cogn. Comput. (ICCI CC)*, London, U.K., 2014, pp. 100–108.

[103] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting DNS amplification attacks," in *Proc. Int. Workshop Crit. Inf. Infrastruct. Security*, 2007, pp. 185–196.

[104] V. Pappas, D. Massey, and L. Zhang, "Enhancing DNS resilience against denial of service attacks," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Edinburgh, U.K., 2007, pp. 450–459.

[105] H. Ballani and P. Francis, "Mitigating DNS DoS attacks," in *Proc. 15th ACM Conf. Comput. Commun. Security*, 2008, pp. 189–198.

[106] L. Zhu *et al.*, "T-DNS: Connection-oriented DNS to improve privacy and security (poster abstract)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 379–380, 2015.

[107] F. Guo, J. Chen, and T.-C. Chiueh, "Spoof detection for preventing DoS attacks against DNS servers," in *Proc. 26th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Lisbon, Portugal, 2006, p. 37.

[108] S. Rastegari, M. I. Saripan, and M. F. A. Rasid, "Detection of denial of service attacks against domain name system using neural networks," *Int. J. Comput. Sci. Issues*, vol. 6, no. 1, pp. 23–27, Nov. 2009.

[109] E. Y. Chen, "Detecting DoS attacks on SIP systems," in *Proc. 1st IEEE Workshop VoIP Manag. Security*, Vancouver, BC, Canada, 2006, pp. 53–58.

[110] D. Geneiatakis, N. Vrakas, and C. Lambrinoudakis, "Utilizing bloom filters for detecting flooding attacks against SIP based services," *Comput. Security*, vol. 28, no. 7, pp. 578–591, 2009.

[111] H. Sengar, H. Wang, D. Wijesekera, and S. Jajodia, "Fast detection of denial-of-service attacks on ip telephony," in *Proc. 14th IEEE Int. Workshop Qual. Service (IWQoS)*, New Haven, CT, USA, 2006, pp. 199–208.

[112] J. Tang and Y. Cheng, "Quick detection of stealthy SIP flooding attacks in VoIP networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kyoto, Japan, 2011, pp. 1–5.

[113] D. Allawi, A. A. Rohiem, A. El-Moghazy, and A. Ghalwash, "New algorithm for SIP flooding attack detection," *Int. J. Comput. Sci. Telecommun.*, vol. 4, no. 3, pp. 10–19, 2013.

[114] J. Tang, Y. Cheng, Y. Hao, and W. Song, "SIP flooding attack detection with a multi-dimensional sketch design," *IEEE Trans. Depend. Secure Comput.*, vol. 11, no. 6, pp. 582–595, Nov./Dec. 2014.

[115] D. Golait and N. Hubballi, "Detecting anomalous behavior in VoIP systems: A discrete event system modeling," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 3, pp. 730–745, Mar. 2017.

[116] A. Kumar and S. Tilagam, "A novel approach for evaluating and detecting low rate SIP flooding attack," *Int. J. Comput. Appl.*, vol. 26, no. 1, pp. 31–36, 2011.

[117] M. Semerci, A. T. Cemgil, and B. Sankur, "An intelligent cyber security system against DDoS attacks in SIP networks," *Comput. Netw.*, vol. 136, pp. 137–154, May 2018.

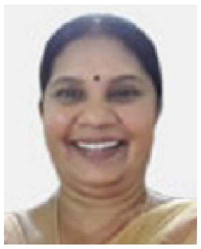
[118] I. Hussain and F. Naït-Abdesselam, "Strategy based proxy to secure user agent from flooding attack in SIP," in *Proc. 7th IEEE Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Istanbul, Turkey, 2011, pp. 430–435.

[119] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Security*, vol. 31, no. 3, pp. 357–374, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404811001672

[120] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on Web servers in the presence of sampling," *Comput. Netw.*, vol. 121, pp. 25–36, Jul. 2017.

[121] J. Stanek and L. Kencl, "SIPp-DD: SIP DDoS flood-attack simulation tool," in *Proc. 20th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2011, pp. 1–7.

Amit Praseed received the B.Tech. degree in computer science and engineering from the College of Engineering Trivandrum, Kerala, India, in 2014, the M.Tech. degree in computer science and engineering (information security) from the National Institute of Technology Karnataka, Surathkal, India, in 2016, where he is currently pursuing the Ph.D. degree in computer science. His research interests include Web security and information security.



P. Santhi Thilagam received the B.E. degree in computer science and engineering from the Thiagarajar College of Engineering, Madurai, the M.E. degree in computer science and engineering from Anna University, Chennai, India, in 2000, and the Ph.D. degree in computer science and engineering from the National Institute of Technology Karnataka (NITK), Surathkal, India, in 2008, where she has been with the Department of Computer Science and Engineering, since 1996, and was an Assistant Professor, and became an Associate Professor in 2009. Since 2018, she has been a Professor with NITK. Her current research interests include cloud computing, distributed data mining, and Web security. She was a recipient of the BITES Best Ph.D. Thesis Award for the year 2009 in Computer Science and Engineering Category.