Very Short Intermittent DDoS Attacks in an Unsaturated System

Huasong Shan[1], Qingyang Wang[1(✉)], and Qiben Yan[2]

[1] Louisiana State University, Baton Rouge, LA 70803, USA
{hshan1,qwang26}@lsu.edu
[2] University of Nebraska-Lincoln, Lincoln, NE 68588, USA
qyan@cse.unl.edu

Abstract. We present a new class of low-volume application layer DDoS attack–Very Short Intermittent DDoS (VSI-DDoS). Such attack sends intermittent bursts (tens of milliseconds duration) of legitimate HTTP requests to the target website with the goal of degrading the quality of service (QoS) of the system and damaging the long-term business of the service provider. VSI-DDoS attacks can be especially stealthy since they can significantly impair the target system performance while the average usage rate of all the system resources is at a moderate level, making it hard to pinpoint the root-cause of performance degradation. We develop a framework to effectively launch VSI-DDoS attacks, which includes three phases: the profiling phase in which appropriate HTTP requests are selected to launch the attack, the training phase in which a typical Service Level Agreement (e.g., 95^{th} percentile response time <1 s) is used to train the attack parameters, and the attacking phase in which attacking scripts are generated and deployed to distributed bots to launch the actual attack. To evaluate such VSI-DDoS attacks, we conduct extensive experiments using a representative benchmark web application under realistic cloud scaling settings and equipped with some popular state-of-the-art IDS/IPS systems (e.g., Snort), and find that our attacks are able to effectively cause the long-tail latency problem of the benchmark website while escaping the radar of those DDoS defense tools.

Keywords: Long-tail latency · Performance bottleneck
n-tier systems · Pulsating attack · Web attack · DDoS attack

1 Introduction

Distributed Denial-of-Service (DDoS) attacks for Internet services such as social networks and e-commerce are increasing in sophistication and scale. Kaspersky Lab's "DDoS Intelligence Report Q1 2017" [4] reports that the trend of DDoS attacks has been increasing despite numerous DDoS defense mechanisms. One important reason of the increasing popularity of DDoS attacks is due to the ever-evolving new types of DDoS attacks that exploit various newly discovered

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 45–66, 2018.
https://doi.org/10.1007/978-3-319-78813-5_3

network or system vulnerabilities, bypassing the state-of-the-art defense mechanisms [30,38]. The damage that these DDoS attacks cause to enterprise organizations is well-known, and includes both monetary (e.g., $40,000 per hour) and customer trust losses [15]. Therefore, for guarding these Internet services, it is very important to detect, prevent and mitigate various emerging DDoS attacks.

In this work, we present a new low-volume application-layer DDoS attack called Very Short Intermittent DDoS (VSI-DDoS). A VSI-DDoS attacker sends intermittent bursts of carefully chosen legitimate HTTP requests to the target system, with the aim of creating "Unsaturated DoS", where the denial of service can successfully last for short periods of time (i.e., hundreds of milliseconds). VSI-DDoS attacks are not to bring down the system as traditional flooding DDoS attacks do, but rather to degrade the quality of service by causing frequent and sometimes intolerable delays for legitimate users, which will eventually damage the long-term business goal of the target system. For example, given that modern web applications care more about the tail latency than the average latency [12] (e.g., Google requires 99% of its web-search to finish within 0.5 s [13]), a long-tail latency (e.g., 95^{th} percentile response time >1 s) caused by a VSI-DDoS attack can significantly affect the target website's business and reputation.

Compared to previous research on network-layer pulsating DDoS attacks [17,18,22,23,25,27], VSI-DDoS is a type of application-layer DDoS attacks, with even lower level of traceability and better stealthiness. Unlike the network-layer pulsating attacks which intend to temporarily saturate the bandwidth of network links that connect to the target system, VSI-DDoS attacks aim to create very short saturations of the bottleneck resource (usually in CPU or disk I/O) inside the target system, which we refer to as very short bottlenecks (VSBs) and typically require much less amount of attack traffic to trigger them. Less amount of attack traffic leads to a higher level of stealthiness. In addition, a VSI-DDoS attack adopts legitimate HTTP requests, which can easily penetrate the defense mechanisms adopted by CDNs, network routers or switches in the path to the target system, thereby reducing the detection surface.

To effectively mount VSI-DDoS attacks, we should fully understand the triggering conditions of VSBs inside the target system, and quantify their long-term damages on the overall system performance. We develop a three-phase framework to tackle these challenges, which involves profiling, training, and attacking. Specifically, in the profiling phase we profile all the HTTP requests supported by the target website and select a set of appropriate ones to launch the attack. We find that heavy requests (e.g., the request with long service time consuming more bottleneck resource in the target web system) can achieve significantly better attack efficiency than light requests; only a small burst of heavy requests are needed to trigger VSBs of the target system, reducing the cost of an effective attack. In the training phase we use a typical Service Level Agreement (e.g., 95^{th} percentile response time <1 s) for most e-commerce websites as an evaluation metric to train the key parameters of an effective VSI-DDoS attack, including burst volume, length, and interval. We find that an appropriate combination of these parameters not only achieves high attacking efficiency, but also escapes

the radar of the most popular state-of-the-art DDoS defense tools, which further validates the stealthiness of the proposed attack.

In summary, the main two contributions of this work are:

- We present a novel low-volume application-layer VSI-DDoS attack that can broadly threaten a wide range of web applications in a stealthy manner. Unlike the traditional brute-force DoS attacks or pulsating attacks which focus on network bandwidth, VSI-DDoS attacks target the bottleneck resource of the target web system using legitimate HTTP requests, thereby reducing the cost of an effective attack while keeping the attack highly stealthy.
- We develop a three-phase framework via an empirical approach that is able to efficiently launch VSI-DDoS attacks against a target web application. Through a representative web application benchmark under realistic cloud scaling settings and equipped with the most popular state-of-the-art DDoS defense tools, we validate the practicality of our attacking framework.

Through our evaluation of VSI-DDoS attacks under realistic cloud scaling settings and IDS/IPS systems, we confirm that the proposed attacks not only bypass the triggering conditions of the cloud scaling but also invalidate capacity-based threshold monitoring and detection. We further explore two more potential solutions to VSI-DDoS attacks: fine-grained VSBs detection and user behavior model validation, and discuss their strengths and weaknesses in practice.

The remainder of this paper is organized as follows. Section 2 presents the origin and motivation of VSI-DDoS attacks. Section 3 describes the definition of VSI-DDoS attacks, and the design of the VSI-DDoS attack framework. Section 4 evaluates the effectiveness and stealthiness of our attacks. Section 5 discusses some countermeasures and future work. Section 6 presents the related work and Sect. 7 concludes the paper.

2 Background and Motivations

2.1 Origin of VSI-DDoS Attacks

VSI-DDoS attacks originate from the new phenomenon of very short bottlenecks (VSBs), also called transient bottlenecks in recent performance analysis of Internet services deployed in Cloud environments [35,36]. In these previous studies VSBs have been identified as one of the main sources for the puzzling performance anomalies of the cloud-host web applications even though the system is far from saturation. From time to time cloud practitioners have reported that n-tier web applications produce very long response time (VLRT) requests on the order of several seconds, when the system average utilization is only about 50–60%. The VLRT requests themselves do not contain bugs, since the same requests return within tens of milliseconds when no bottleneck exists in the target system. The reason why VSBs can turn these normal short requests into VLRT requests is because VSBs can cause a large number of requests to queue

in the system within a very short time. Due to some system level concurrency constrains (e.g., limited threads) of component servers, additional requests that exceed the concurrency limit of any component server will be dropped, causing TCP retransmissions (minimum time-out is 1 s [19]). The requests encountered TCP retransmissions become VLRT requests perceived by the end users.

While most of previous studies focus on VSBs caused by internal system level factors such as Java garbage collection, CPU Dynamic Voltage and Frequency Scaling (DVFS), interference of collocated virtual machines, this newly identified system vulnerability (VSBs) also motivates us to study the hypothetical VSI-DDoS attacks. Our hypothesis is that the external burst of legitimate HTTP requests can cause erratic fluctuation of resource consumption to be injected into the target system and cause VSBs in the weakest node of the whole distributed system, which in turn cause queue overflow and VLRT requests resulting from TCP retransmissions. Such VSI-DDoS attacks can potentially impose significant threats on current cyber infrastructures while remaining stealthy under the radar of state-of-the-art DDoS defense mechanisms and IDS/IPS systems.

2.2 Importance of Tail Latency

In web applications such as e-commerce, rapid responsiveness is vital for service providers' reputation and business. For example, Google requires 99% of its web-search to finish within 0.5 s [13]; Amazon reported that an every 100 ms increase in the web-page load reduces sales by 1% [24]. In practice, the tail latency, instead of the average latency, is of special concern for mission-critical web-facing applications [12–14,20]. In shared infrastructures such as cloud environments, service level agreements (SLAs) are commonly used for specifying desirable response times, typically within one or two seconds [12]. In this case, only requests with response time within the specified threshold have a positive impact to service providers' business, and the requests with long response time (beyond the threshold), not only waste network and system resources, but also cause penalties (negative impact in revenue) to the business of the service provider. In general, 99th, 98th, and 95th percentile response time are representative metrics to measure the performance of web applications [12,26]. In this paper, we also use percentile response time as the evaluation metric to measure the effectiveness of an adversary's VSI-DDoS attacks.

2.3 Measured Long-Tail Latency Caused by VSI-DDoS Attacks

Here, we show the impact of VSI-DDoS attacks through concrete benchmark results. The benefit of benchmark experiments is to have a fully controlled system, which enables a detailed study about how the target system behaves when it is under a VSI-DDoS attack. The design of the VSI-DDoS attack framework and the real production setting evaluation are in Sects. 3 and 4, respectively.

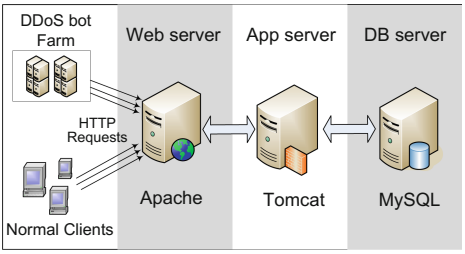
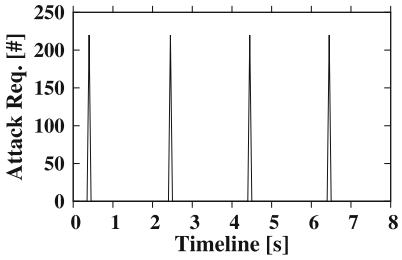
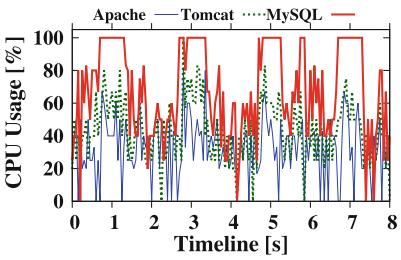


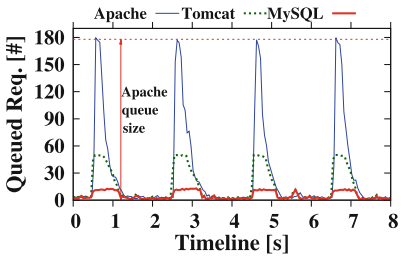
Fig. 1. Experimental sample topology



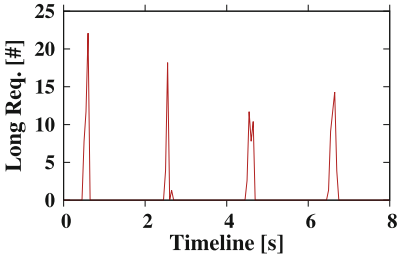
(a) Bursts of attacking HTTP requests in a VSI-DDoS attack during an 8-second time period. We count the # of attacking requests in every 50ms time window.



(b) Transient CPU saturations in MySQL coincide with bursts of attacking requests in (a), suggesting that the VSI-DDoS attack creates frequent VSBs in MySQL.

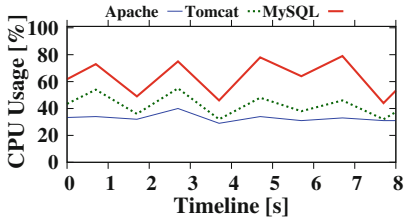


(c) VSBs in MySQL (see (b)) cause requests to queue in MySQL, which in turn push requests to queue in upstream Tomcat and Apache. The horizontal line shows the queue limit in the front-most Apache; once exceeded, new requests are dropped.

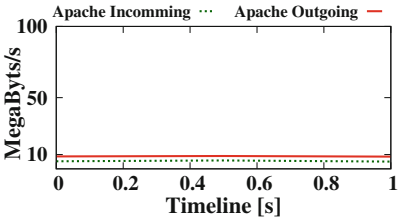


(d) Number of requests (from legitimate users) with response time > 1s during the same 8-second time period. Such long requests are caused by TCP retransmissions once the front-most Apache drops incoming requests (see (c)).

Fig. 2. Measured performance of the benchmark application under a VSI-DDoS attack. Bursts of attacking HTTP requests (a) trigger VSBs in the bottom-most MySQL of the system (b), which cause requests to queue from local to the front-most Apache (c). Queue-overflows occur in Apache, causing TCP retransmissions and long response time requests (d). (Color figure online)



(a) CPU util. of each server during the same 8s attacking period as in Fig. 2.



(b) Incoming/outgoing network traffic of Apache. The bandwidth is 1 Gbps.

Fig. 3. Resource utilization of each server in the system under the VSI-DDoS Attack. Metrics are measured using *vmstat* at every 1 s. (a) and (b) show that both CPU and network bandwidth are not saturated. Utilization of other resources (e.g., memory) are omitted since they are far from saturation.

Benchmark Measurements. We use RUBBoS [6], a representative n-tier web application benchmark modeling the popular news forum website such as *Slashdot*. In Fig. 1, we show the basic configuration for RUBBoS using the typical 3-tier architecture, with 1 Apache web server, 1 Tomcat application Server, and 1 MySQL database server deployed in an academic cloud platform (more details of experimental setup in Sect. 4.1). RUBBoS can emulate the behavior of legitimate users to surf the website. Each user follows a Markov chain model to navigate among different webpages, with averagely 7-s think time between receiving a web page and submitting a new page request. On the other hand, we adopt *Apache Bench* to send intermittent bursts of carefully chosen legitimate HTTP requests; each burst is injected within a very short time window (e.g., 50 ms).

The mechanism of how VSI-DDoS attacks impact the performance of the target n-tier web system can only be seen using fine-grained monitoring. Figure 2 shows such an analysis when the target 3-tier benchmark website serving 3000 legitimate users is under a VSI-DDoS attack. All the metrics in the subfigures are measured at every 50 ms time window. Figure 2a shows that the burst of attacking requests occurs in every 2 s. Each burst contains about 250 legitimate HTTP requests supported by the benchmark website within a 50 ms time window. The bursts of attack requests cause transient CPU saturations of MySQL in Fig. 2b. These transient CPU saturations create VSBs and cause requests to queue in MySQL; MySQL local queue soon fills up (at 0.5 s, 2.5 s, 4.5 s, and 6.5 s), pushing requests to queue in upstream Tomcat and Apache in Fig. 2c. We call this phenomenon as *push-back wave*. Once the queued requests in the frontmost Apache exceed its queue limit (180 in our configuration), new requests from legitimate users will be dropped, leading to TCP retransmissions and very long response time (VLRT) requests as we observed in Fig. 2d.

We also find that the VSBs cannot be observed by our normal system monitoring tools (e.g., *sar, vmstat, top*) with the typical 1-s monitoring granularity. Figure 3a shows that the CPU utilization of each server in the system is not saturated all the time using *vmstat* during the 8-s VSI-DDoS attacking period. Figure 3b shows the outgoing/incoming network traffic of Apache is at very low rate (<10 MBps). We omit the graphs of resource utilization of other resources (e.g., memory, disk I/O) since all of them are far from saturation. Given such monitoring data, it is difficult for system administrators to trace the cause of the performance problem.

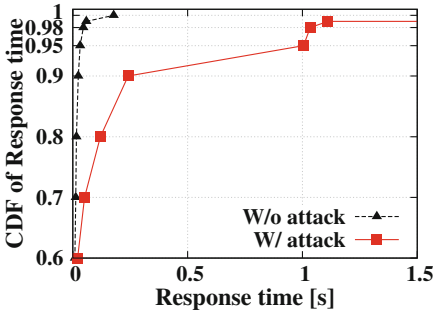


Fig. 4. Percentile response time of the system serving 3000 legitimate users without and with the attack. The 95th percentile response time under attack is >1 s, clearly showing the long-tail latency problem caused by the VSI-DDoS attack. (Color figure online)

To illustrate the negative impact of the VSI-DDoS attack, we compare the percentile response time serving 3000 concurrent legitimate users by the target system under attack and without-attack in Fig. 4. The percentile response time of the system under attack (the red line) uses the same dataset in Fig. 2. This figure shows that all the requests finish within 200 ms without attack (the black line). However, in the attacking scenario, the 95th percentile response time of the target system already exceeds 1 s, clearly showing the long-tail latency problem caused by the VSI-DDoS attack. Such long-tail latency problem is regarded as severe performance issue by most web applications, especially modern e-commerce (e.g., Amazon), as we have introduced in Sect. 2.2.

3 VSI-DDoS Attacks

In this section, we first formally present the adversary's goal of VSI-DDoS attacks and then discuss the key technical challenges of effectively launching VSI-DDoS attacks.

3.1 Goals and Assumptions

In a VSI-DDoS attack scenario the adversary is to create frequent VSBs in the target web system by sending intermittent bursts of legitimate HTTP requests to the target system without being detected. So the goal of VSI-DDoS attacks is not to bring the system down as traditional flooding DDoS attacks do, but rather to degrade the quality of service by causing frequent and sometimes intolerable delays for the legitimate users, which will eventually damage the business of the target system in the long run. Such attacks are stealthy because the target web

system is in an "unsaturated state"; the duration of each created VSB is very short (e.g., 50 ms), which can easily escape the detection of normal monitoring tools adopting coarse granularity statistical analysis (e.g., seconds or minutes).

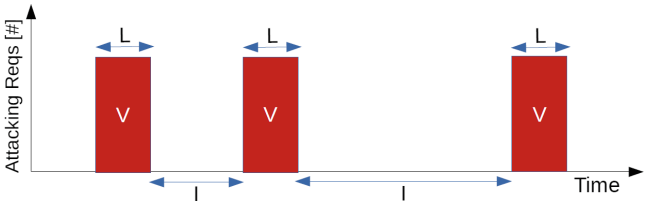


Fig. 5. An illustration of a VSI-DDoS attack, which consists of burst volume V of HTTP requests, burst length L, and burst interval I.

To effectively launch a VSI-DDoS attack, we assume that all the bots under control are coordinated and synchronized so that requests generated by these bots can reach the target web application at the same time or within a very short timeframe. This assumption is reasonable because many previous research efforts already provide solutions, using either centralized [17,33,39] or decentralized methods [23], to coordinate bots to send synchronized traffic and cause network congestion at a specific link. Our focus in this paper is how to create frequent bursts of attacking but legitimate HTTP requests that can effectively trigger VSBs in the target system, causing long-tail latency of the target web system while avoiding being detected. We formally propose VSI-DDoS attacks as follows (Fig. 5):

$$\mathit{Effect} = \mathbb{A}(V, L, I) \tag{1}$$

where,

- *Effect* is the measure of attacking effectiveness; we use percentile response time as a metric to measure the tail latency of the target web system (e.g., 95th percentile response time >1 s). *Effect* is a function of V, L, I.
- V is the number (volume) of attack requests per burst. V should be large enough to temporarily saturate the bottleneck resource in the target system and trigger VSBs. At the same time, V should be small enough to bypass the state-of-the-art threshold-based detection tools [30,38].
- L is the length of each burst. The total requests per burst V will be sent out during the period L. Thus the instant request rate to the target website during a burst period is V/L. L should be short enough to guarantee high instant request rate to trigger VSBs in the target system. Contrarily, too short L will cause large portion of attack requests dropped by the target system due to instant queue overflow (too high V/L), without causing any damage to the target system performance.

– I is the time interval between every two consecutive bursts. I infers the frequency of bursts of HTTP requests to the target system. I should be short enough so that the attacker can generate bursts of HTTP request frequent enough to cause significant performance damage on the target system. On the other hand, too short I makes the attack similar to the traditional flooding DDoS attacks, which can be easily detected.

We note that all the three components need to be carefully coordinated and tuned in order to launch an effective VSI-DDoS attack. To evaluate the effectiveness of such an attack, we measure the tail latency of the target website. We assume that the attack achieves its goal if the measured percentile response time under attack exceeds the predefined threshold, which depends on the SLAs of the target website. Based on this evaluation criteria, we develop an attacking framework which is able to estimate an optimal value of each parameter using an *empirical approach* for an effective VSI-DDoS attack in the following subsection.

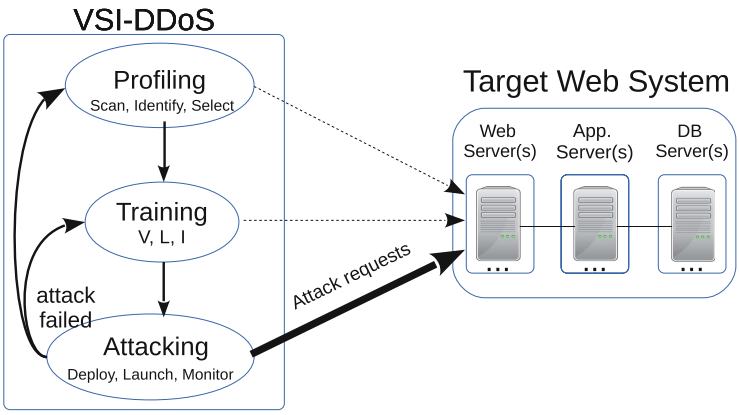


Fig. 6. VSI-DDoS attacks framework

3.2 VSI-DDoS Attack Framework

The proposed VSI-DDoS attack framework contains three phases: profiling, training, and attacking (Fig. 6). The profiling and training phases are to determine the three parameters of a VSI-DDoS attack. The attacking phase generates and deploys attacking scripts to distributed bots and launches the actual attack.

Profiling Phase. This phase selects appropriate types of HTTP requests for attacking in order to create VSBs in the target website with the minimum cost, meaning the least number of attacking requests for each burst. This phase includes the following three steps.

(1) Scanning the supported HTTP requests. To select appropriate HTTP requests, one challenge is to retrieve representative HTTP requests that cover all the transaction types supported by the target website, including both static and GET/POST dynamic requests. Although the requests for static content of a website are easy to retrieve by using some crawling tools (e.g., scrapy), requests for dynamic content are more difficult to get. This is because POST requests are sent out by submitting forms (content is in the body section of a HTTP request), not through the direct URLs. To solve this issue, we adopt a script-based open source web browser *PhantomJS* to retrieve and analyze the form tags inside the HTTP response for every HTTP request. After the attacker provides some initial values for associated input boxes (e.g., user-name and password) inside each form, PhantomJS can submit POST requests automatically. PhantomJS also supports cookies which allows an attacker to conduct consecutive interactions with some websites (e.g., Facebook) which require user login before further website navigation. POST requests are important types of attack requests because they can penetrate Content Delivery Networks (CDN) and attack the original target website. CDNs are widely used by websites nowadays to improve the website performance by caching static content. Since POST requests are dynamic requests which typically require to retrieve/write dynamic information from/to the back-end database, current CDN vendors usually do not support caching responses for POST requests [29]. Thus POST requests are natural candidates to launch effective VSI-DDoS attacks for websites with CDN support.

(2) Identifying *heavy* requests using service time. Once we get enough supported HTTP requests, the next challenge is to decide which requests consume more bottleneck resource (e.g., Database CPU) of the target web system than the others. We term the requests heavily consuming the bottleneck resource as *heavy* requests (e.g., POST requests), meanwhile, those consuming no or little bottleneck resource as *light* requests (e.g., static requests). In this case, *heavy* requests are natural candidates to launch a VSI-DDoS attack because a fewer number of them are needed to trigger VSBs in the target web system than that of light requests. A low number of attacking requests per burst also make the attack stealthy because of the low volume of network traffic. The key question is how do we determine which requests are heavy and which are light?

We use the service time of each type of HTTP requests as a key metric to distinguish the heavy requests from the light ones. Service time of a HTTP request is the time serving the request by the target web system without any queuing delay. Previous research results [35] show that the predominant part of the service time of a request is spent on the bottleneck resource in the system. When the target system is at low utilization[1], service time can be estimated to be the end-to-end response time of a request subtracting the network latency between the client and the target web application. The end-to-end response time of a HTTP request can be easily recorded using *Apache Bench* or *PhantomJS*.

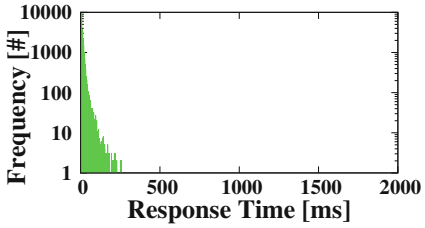
[1] Low utilization is to rule out the queueing effect inside the target system.

Lots of tools can be used to measure the network latency (e.g., the *ping* command). We measure the service time of each type of HTTP requests multiple times and employ the average in order to mitigate influence of network latency variation.

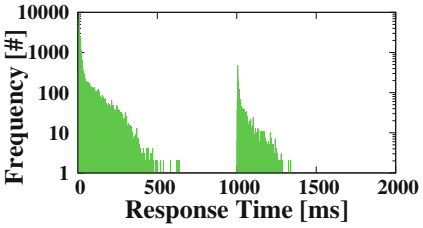
(3) Selecting candidate requests. A naive strategy to select candidate attacking requests is always choosing the heaviest type of requests. In this case, the attacker can use the minimum number of requests per burst to create VSBs in the target web system. However, single type of attacking requests in a VSI-DDoS attack has the risk to be identified as abnormal by existing DDoS defense tools using statistical analysis. For example, the defense tool may simply aggregate all the requests sent out from the same IP and identify that some IPs only send one type of HTTP requests, which is highly suspicious. To bypass such statistics-based detection mechanisms [7], an attacker can select a set of top-ranked heavy requests, which can achieve the same attacking goal with slightly increased cost. Some more advanced defense mechanisms use machine-learning based techniques to learn a legitimate user behavior model [32] and infer suspicions requests if the sequence or the transition probability among them significantly deviates from the model (judging based on pre-defined thresholds). In this case the attacker needs to select candidate requests more carefully to make sure that the sequence of HTTP requests sent from a bot is feasible for a legitimate user. We will discuss this in more detail in Sect. 5.

Training Phase. This phase is to train the key parameters (V, L, and I) of an effective VSI-DDoS attack that meets the adversary's goal (e.g., 95th percentile response time >1 s).

(1) Training volume. The technical objective of a VSI-DDoS attack is to create frequent VSBs in the target web system. Thus a key challenge of VSI-DDoS attacks is to determine whether a batch of attacking requests are able to create a VSB in the target system or not. In most cases the attacker has no privilege to monitor the resource utilization of the target system. Thus the attacker cannot depend on internal resource monitoring to determine the occurrence of VSBs. However, we know that the occurrence of a VSB will create temporary request congestion inside the target system; once the queued requests exceed any system-level queue capacity (e.g., thread pool size), new arriving requests will be dropped and TCP retransmissions (minimum time-out is 1 s) will happen, leading to long response time perceived by the end users (see Fig. 2). In this case, long response time caused by queue-overflow and TCP retransmissions can be treated as a signal of the occurrence of VSBs. Given such an idea, a VSI-DDoS attacker can gradually increase the volume of the attacking requests per batch until the observation of requests with abnormally long response time.



(a) 20 requests per burst case. The low response times of all the requests indicate no TCP retransmission.



(b) 100 requests per burst case. The observed response time over 1s is a signal of the occurrence of TCP retransmissions.

Fig. 7. Training the lower bound of volume for an effective VSI-DDoS attack for our RUBBoS benchmark application. We increase the volume of attacking requests per burst step by step until we observe abnormally long response time of requests sent from legitimate users (see Fig. 7b).

The minimum volume per burst that triggers the long response time requests due to TCP retransmissions is the lower bound V_{min} for the selected attack requests. Figure 7 shows the process of training V_{min} for an effective VSI-DDoS attack for our RUBBoS website, which is serving 3000 legitimate users. When burst volume is only 20, the response time perceived by all legitimate users is lower than 250 ms in Fig. 7a. We increase burst volume step by step (e.g., 10 or 20) until we observe the requests sent from legitimate users experience abnormally long response time in Fig. 7b. The distinct two-modal response time distribution indicates that 100 reaches the lower bound. In practice we set the volume higher than V_{min} to guarantee the successful triggering of VSBs in the target system and achieve better attack result as shown in Fig. 8. On the other hand, the volume should not be too high otherwise it will trigger the alarm of defense tools (e.g., Snort [7]) deployed in the target web system. We can increase the number of bots and reduce the number of attack requests per bot to bypass the state-of-the-art detection mechanisms.

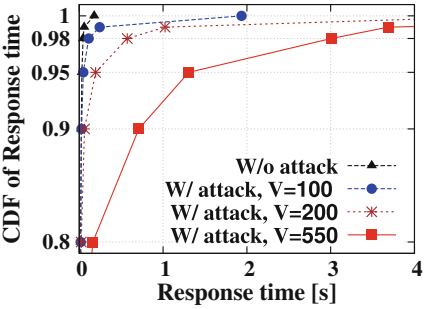


Fig. 8. Impact of volume V per burst. This figure shows the percentile response time of the target system (serving 3000 legitimate clients) gets more damages as the attack volume per burst V increases from 100 to 550 (fixing $L = 50$ ms, $I = 7$ s).

(2) Training burst length. A good burst length L should maximize the impact of the burst of attacking requests on the requests sent from legitimate users. We observed that the best L should be the service time of the selected attacking requests. A HTTP request that originates from a client arrives at the web server, which distributes it among the application servers, which in turn ask the

database to execute the query. Due to the RPC-style synchronous communication between consecutive tiers, the processing threads and other associate soft resources such as database connections of a component server will be occupied until all the activities in the downstream tiers are done. In order to create the most soft resource consumption, the burst of attacking requests should arrive within the service time of the attacking requests. In this case, all the attacking requests will stay in the target system before any of them finishes processing and leaves the system. Once any of soft resources in any tier of the system are exhausted, new requests from legitimate users will be dropped, leading to long response time requests resulting from TCP retransmissions.

Figure 9 shows the impact of the burst length L on the tail latency of our RUBBoS website. We choose the heavy request "ViewStory" as the attacking requests. The service time of such heavy request is about 50 ms. Note that the biggest attacking damage appears when L is 50 ms. Too short L leads to low effectiveness for the attacking burst, since most of the attacking requests will be dropped due to sudden queue-overflow[2], without causing any harm to the requests from legitimate users. On the other hand, too long L (e.g., the $L = 800$ ms case) leads to low instant request rate (V/L), which may not be able to create VSBs in the target system and lead to inferior attacking results.

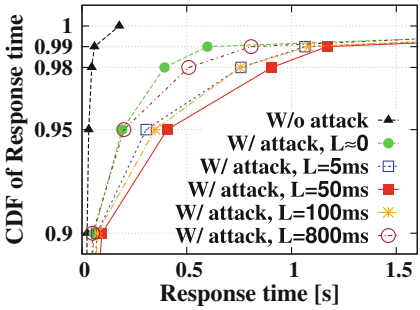


Fig. 9. Impact of the length L per burst. The biggest attacking damage (serving 3000 legitimate clients, and fixing $V = 200$, $I = 7$ s) appears when L is 50 ms; the more the burst length deviates from 50 ms, the weaker the damage caused by the VSI-DDoS attack is.

(3) Training interval between bursts. By determining V and L of each burst we can make sure one burst is able to trigger a VSB in the target system. The final goal of a VSI-DDoS attack is to create the long-tail latency problem of the target web system. Too small interval between bursts makes it similar as the traditional flooding DDoS attack, thus can be easily detected. Too large interval creates insufficient number of VSBs in the target web system, thus unable to achieve the adversary's goal. To select a reasonable interval, we start from a relatively large interval and gradually reduce the interval until the measured tail latency meets the adversary's goal. Figure 10 shows such a process of selecting a reasonable interval for our RUBBoS benchmark. To avoid an obvious burst pattern of attacking requests, the interval between consecutive bursts is not necessarily assigned with a fixed value. A VSI-DDoS attacker can design the interval with a random variable following certain statistical distributions, with the mean

[2] Short L leads to high instant request rate V/L, OS kernel may not be able to handle packets promptly due to high overhead of interrupt handling [18].

to be similar to a normal user's think time between consecutive requests. Figure 2 in Sect. 2.3 is such an example. The interval between attack bursts follows a normal distribution with the mean of 2 s.

Attacking Phase. This phase is to launch the real VSI-DDoS attack based on the previous profiling and training results of V, L, I for request bursts. Attackers launch the attack by leveraging the resources of multiple hosts, especially Botnet. We note that an attacker should use a probe to continuously monitor the performance of the target website, for example, send a sequence of very light HTTP requests (e.g., html) at regular intervals and check the response time distribution. The profiling and training phase need to redo once the attacking results cannot meet the adversary's goal due to the change of baseline workload or system state (e.g., dataset size change).

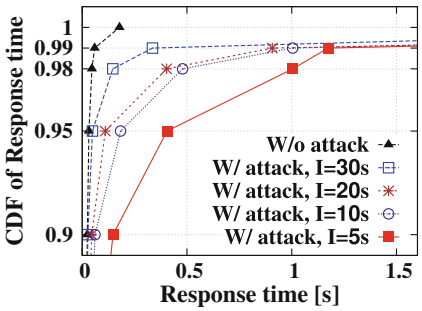


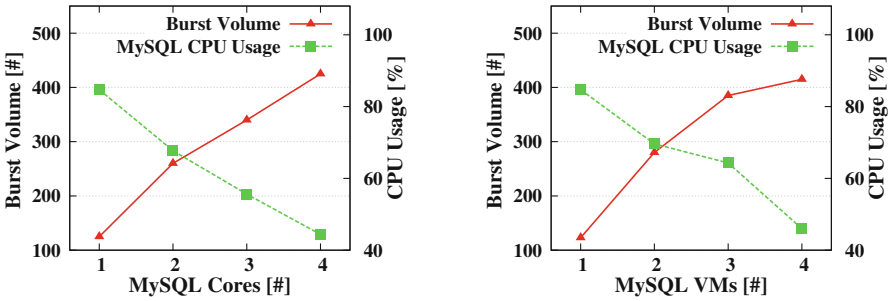
Fig. 10. Impact of the interval I between bursts. This figure shows the percentile response time of the system (serving 3000 legitimate clients) gets more damages as we decreases I from 30 s to 5 s (fixing $V = 200$, $L = 50$ ms).

4 Evaluation

4.1 VSI-DDoS Attacks Under Cloud Scaling

To evaluate the effectiveness of our VSI-DDoS attacks in the real production settings, we deploy RUBBoS in a popular NSF sponsored cloud platform-Cloudlab [5].

Experiment Methodology. In the real production environment, once administrators pinpoint the performance bottleneck of an n-tier system, they can solve the issue by scaling the bottleneck tier. One policy is scaling up (updating the hardware of the bottleneck tier), and the other is scaling out (adding more machines/virtual machines to the bottleneck tier). For example, Amazon Auto Scaling [1] can scale out EC2 instances as the demand of an application increases. We evaluate our attack under both scaling settings. In our experiments, we assume that the bottleneck is MySQL since the bottleneck typically takes place in the database due to the high resource consumption of database operations. To evaluate our attack under the cloud scaling settings, we keep all the software configuration (e.g., queue size, DB connection pool size) the same to rule out their impacts to our evaluation. In the scaling up case, we update the hardware unit (1 CPU core and 1 GB Memory) of MySQL from 1 to 4 Units. In the scaling out case, we increase the number of MySQL VMs from 1 to 4. All the



(a) Scaling CPU cores of MySQL. (b) Scaling Virtual Machines of MySQL.

Fig. 11. The required burst volume and the CPU utilization of MySQL in scaling up/out scenarios while achieving our victim goal. Average CPU Usage decreases after scaled, indicating the scaling mitigates the bottleneck. However, we can still get the attacking goal by increasing burst volume.

VMs (Xen-based Emulab virtual nodes) are running CentOS 6.5 on 2.10 GHz Intel Xeon E5-2450 processors. To maximize the impact of our attack, we set the burst length as the service time of attack requests (e.g., 50 ms.), and set the burst interval as 2 s. We conduct the attack experiments for 10 min in each scenario since half of the DDoS attacks last longer than 10 min [31]. Our DDoS bot farm in Fig. 1 consists of 8 machines, one serves as a centralized controller that coordinates and synchronizes the other nodes to launch the attacks.

Results. Figure 11 depicts the required burst volume and the relevant usage of the bottleneck resource at scaling up/out scenarios to achieve our attacking goal. We can see the average CPU utilization of MySQL reduces from high load (>80%) to moderate level (<50%) after the bottleneck tier is scaled, indicating the scaling policies are effective since more CPU cores or VMs can mitigate the impact of the bottleneck resource to the system performance. However, we can still reach our attacking goal by increasing attacking volume per burst even in a large scale scenario, since it requires higher burst volume to trigger VSBs in the system after the capacity of the bottleneck tier increases. On the other hand, increasing the attack requests by each bot obviously increases the risk of detection by the target system, but we can coordinate more synchronized bots to send higher volume per burst to achieve our attacking goal using decentralized synchronization mechanism [23]. As such, we can still keep our attacks under the radar of the state-of-the-art detection mechanism.

Remarks About Cloud Scaling. In real production clouds (e.g., Amazon AWS), the users can customize some triggering conditions to instruct Amazon Auto Scaling [1] to scale out/in instances in response to metrics (such as bandwidth usage or CPU utilization) monitored by Amazon CloudWatch [2]. The monitoring granularity of Amazon CloudWatch for premium users is 1 min [2].

For example, the target system can add more VMs once the average CPU utilization of all the instances exceeds 85% during a 1-min statistical period. From Sampling Theory, our attack is hard to trigger the scaling plans sampling in minutes level, since the VSBs usually occur in milliseconds level (more details about monitoring granularity in Sect. 5). Large-scale web applications typically adopt dynamic scaling strategy for better resource efficiency and load balancing, VSI-DDoS attacks can avoid the triggering conditions of the cloud scaling, thus we expect that the current cloud scaling techniques do not help resolving our attacks.

4.2 VSI-DDoS Attacks Under Defense Tools

To validate the stealthiness of our attacks under the popular defense mechanisms, we deploy some defense tools before the web tier in our RUBBoS environments.

Experiment Methodology. Snort is the most widely deployed network anomaly detection system in the world that is acquired by Cisco Systems on October 7, 2013, and widely used in practice for DDoS defense [7,8]. Snort.AD [9], extended based on Snort, is a threshold and statistics-based network anomaly detection tool, which can analyze the network traffic based on different protocols (UDP, TCP, HTTP, etc.) within a certain period. Here, we take HTTP traffic as a representative metric in Snort.AD to evaluate whether our attacks break through the cordon, since our attacking requests only involve the HTTP packets. In the following experiments, we configure 2000 and 4000 concurrent legitimate users as the baseline for low and high background workload scenarios. We set 95th, 98th and 99th percentile response time (>1 s) as the candidate attacking goals, and call them *95th, 98th, and 99th case* hereinafter. To achieve these different attacking goals, we fix the burst length as the service time of the attacking requests and the burst interval as 2 s, and only tune the burst volume ((250, 150, 100) and (150, 100, 50) for the 95th, 98th and 99th case of 2000 and 4000 baseline, respectively). We conduct the experiments in a 10-min period for each scenario. We modify the code of Snort.AD to trace the number of the HTTP incoming/outgoing packets in a minute interval and the HTTP incoming/outgoing speed in terms of Mega Bytes per second, to evaluate whether they exceed the threshold for these cases with different attacking goals and background workload.

Alert Threshold Setting. How to set the alert threshold is a well-known challenge for administrators [10,11]: a high threshold may not be able to detect anomalies; a low threshold may incur a high number of false positive alarms which an administer tries to avoid in practice. Typically, a widely-adopted setting strategy [10,11] is to set the threshold of each monitoring metric based on the capacity of the target system. Network security company [10] recommends that the company's IT team should conduct the necessary performance tests to determine the capacity, and set the threshold lower than the capacity to prevent resource exhaustion (e.g., define the threshold when reaching 85% bottleneck

Table 1. Measured HTTP traffic in the cases of 95th, 98th and 99th percentile response time (>1 s) as candidate attacking goals. All of measured metrics are less than the predefined thresholds set based on system capacity when the corresponding attacking goal is achieved.

Metrics	Threshold	2000 low load				4000 high load			
		95th	98th	99th	B/L	95th	98th	99th	B/L
In. packets (#/min)	299K	158K	119K	111K	99K	224K	214K	208K	201K
Out. packets (#/min)	349K	171K	134K	127K	116K	259K	249K	241K	233K
In. speed (MB/sec)	9.32	4.68	3.96	3.62	3.11	7.08	6.76	6.45	6.23
Out. speed (MB/sec)	17.83	7.62	6.83	6.48	5.94	12.78	12.46	12.12	11.89

In.: HTTP Incoming, **Out.**: HTTP Outgoing, **B/L**: Baseline

resource utilization of the web system). We also profile the capacity of the target system in our experimental environment under a worst-case scenario (when serving 6000 concurrent users, the bottleneck resource, MySQL CPU utilization, reaches 85%). In our experiments, we take this widely-adopted strategy to define the alert threshold listed in Column 2 of Table 1 to capture our attacks.

Results. Table 1 lists the maximal HTTP incoming/outgoing packets and speed under our attacks for the cases with different attacking goals and background workload. All of the measured traffic metrics are in the moderate level and far from the predefined threshold (based on system capacity) when the corresponding attacking goal is achieved, indicating that our attacks create an "Unsaturated illusion" for Snort. As a result, Snort reports no alert. More importantly, the increased traffic due to our attacks is small compared to the baseline case, especially when the attacking goal is less aggressive (e.g., 99th percentile response time >1 s). For example, an effective VSI-DDoS attack in the 99th case only incurs 10% more traffic when the baseline workload is 2000, and 4% more traffic when the baseline is 4000. This result also suggests that an effective VSI-DDoS attack is easier to achieve as the background traffic increases.

Remarks About Threshold-Based Detection. The fundamental reason that our attack (in *milliseconds* level) can invalidate the traditional threshold-based detection tools (in *seconds or minutes* level) is their coarse monitoring granularity. The coarse monitoring granularity is effective for identifying brute-force DDoS attacks and flash crowds lasting for tens of seconds or minutes [21] (detailed explanation in Sect. 6), but obviously too long to observe any abnormal behaviors triggered by a VSI-DDoS attack lasting for only tens of milliseconds (e.g., the minimum measured rate-interval of the Cisco Adaptive Security Appliance is 1 s [3], the minimum sampling interval of Snort is 1 min [9], the sampling interval of BotSniffer's monitor engine [16] is in seconds level). Indeed, fine-grained monitoring could mitigate the problem, but with the cost of high monitoring overhead and potentially high false positive alarms (falsely block legitimate users), because web application workload is naturally bursty [21]. We will discuss the impact of monitoring granularity in more detail in the following.

5 Discussion of Possible Detection/Defense Mechanisms

Here, we introduce two more candidate countermeasures for VSI-DDoS attacks and discuss their pros and cons in practice.

(1) Fine-Grained VSBs Detection. A natural way to detect a VSI-DDoS attack is to detect the occurrence of VSBs in the target web system, and determine whether they are caused by bursts of malicious HTTP requests. However, detecting VSBs in the target web system is challenging because they usually occur in milliseconds level; from Sampling Theory, these VSBs would not be reliably detectable by normal tools sampling at time intervals from 1 s (e.g., *Snort, BotSniffer* [16], *sar, vmstat, top*) to several minutes (e.g, *CloudWatch*). To reliably detect VSBs and their correlation with a potential VSI-DDoS attack, we need both the system and application level fine-grained monitoring (millisecond level). System-level monitoring is to detect VSBs by collecting the hardware resource utilization of all component servers in the target system using fine-grained monitoring tools (e.g., *collectl*). Application level monitoring is to collect the request processing logs of each component server in the system and analyze the performance metrics such as incoming request rate, queue status, and point-in-time response time in fine granularity. Given the collected fine-grained monitoring data, we apply a timeline correlation analysis to link the observed VSBs in system-level monitoring with the application level performance metrics, as we have illustrated in Sect. 2.3 (see Fig. 2). On the other hand, with "coarse" monitoring granularity (e.g., 1 s), these metrics only show moderate variations or non-saturation (see Fig. 3) over time, which will likely not bring any attention to administrators. Although the fine-grained monitoring approach is conceptually simple, it requires sophisticated fine-grained monitoring tools. [36] shows that VSBs can be caused by the temporary saturation of any system resource that is in the execution path as HTTP requests flow via the system. Specifically, VSBs caused by a VSI-DDoS attack may not necessarily be in hardware resources, but in system soft resources (e.g., database locks, thread pool) that are out of the scope of existing fine-grained monitoring tools (e.g. *collectl*). We observed this phenomenon when we deploy Opentaps, a popular open source ERP/CRM web application, in Amazon EC2 cloud platform. The target Opentaps web application shows a significant long-tail latency problem under a VSI-DDoS attack while *collectl* reports no saturation of any hardware resources. In addition, monitoring overhead is another big concern of the fine-grained monitoring approach. In our RUBBoS experiments, we observe that *collectl* incurs high overhead at sub-second sampling intervals (about 6% CPU utilization overhead at 100 ms interval and 12% at 20 ms).

(2) User Behavior Model Validation. Some advanced defense mechanisms use machine-learning based techniques to learn a normal user behavior model from web server logs. These user behavior models [32,37] are used to differentiate HTTP requests sent by humans from those sent by bots. For example,

Oikonomou and Mirkovic [32] model three aspects of human behaviors such as inter-arrival time of consecutive requests from the same user, choice of content to access, and ability to ignore invisible content. Such user behavior models are indeed effective if a VSI-DDoS attacker chooses single type of requests to attack or a set of heavy requests that have low transition probability among them. However, the attacker can set the interval between consecutive bursts in a VSI-DDoS attack similar to the browsing behavior of a legitimate user (e.g., an average 7-s think time between two webpages). The attacker can also learn a popular sequence of HTTP requests that a legitimate user will likely go through when visiting the target website. Then the same sequence of requests can be selected as the attack requests. Such selection strategy may not be the most cost-efficient one since not all of the selected requests are heavy, but the attacker can still achieve the goal by controlling more bots to launch the attack. Defense mechanisms that adopt user behavior models certainly raise the bar of our attacks, but they do not suffice to detect and defend such attacks.

6 Related Work

DDoS attack and defense mechanisms have been extensively investigated and categorized in survey papers [30,38]. In this section, we review the most relevant work in two aspects: the low-rate network-layer pulsating DDoS attacks and the low-volume application-layer DDoS attacks [28].

Low-Rate Network-Layer Pulsating DDoS Attacks. Many attack mechanisms in this category [17,22,23,25,27] have been proposed, which send bursts of TCP packets to cause packet drops, by exploiting deficiencies in TCP retransmission time-out mechanism known as Shrew attack [25], congestion control response mechanism known as Pulsating attack [23,27], or the transients of the system's adaptation mechanisms known as RoQ attack [17]. These attacks share some similar features with VSI-DDoS attacks, such as the low attacking volume in Shrew and Pulsating attack, and the QoS degradation in RoQ attack. However, our work differs from these work in three aspects: (1) all these attacks are network-layer attacks targeting at network links, while our attacks are at the application-layer, exploiting the bottleneck resource (e.g., CPU, I/O) and the complex resource dependencies (e.g., push-back wave [36]) inside the web system; (2) these attacks usually require a fixed or crafted burst interval to synchronize the Retransmission Timeout (RTO) duration, while our attacks are more flexible in selecting burst volume, length and interval, which allows our attack to be even stealthier; (3) our evaluation metric is based on percentile response time, representing real user experience and provider's service level agreements, which has not been used previously to quantify the attack impact.

Low-Volume Application-Layer DDoS Attacks. One class of low-volume application-layer DDoS attacks specifically related to our attacks are called flash crowds [21], which refer to the scenario when thousands of legitimate users

intensively browse an e-commercial website due to a hot event (e.g., Black Friday deals). Previous detection mechanisms are mainly focusing on differentiating traffic from flash crowds created by legitimate users or application-layer DDoS attacks [21,34,37], such as using hidden semi-Markov model [37] and session-level misbehaviors [34] for anomaly detection. VSI-DDoS attacks can be launched with randomized interval and learn from the user behaviors of a legitimate user, which invalidates those user behavior model-based application layer detection mechanisms. More importantly, VSI-DDoS attacks exploit very short bottlenecks (VSBs) as the system vulnerability, VSBs can be much shorter (tens of milliseconds) than the duration of the traditional application-layer flash crowds traffic (tens of seconds or minutes). Thus, the detection mechanisms of identifying DDoS attacks from flash crowds can be defeated by our VSI-DDoS attacks.

7 Conclusions

We presented a new type of low-volume application layer DDoS attack, VSI-DDoS attacks, exploiting a newly discovered system vulnerability (VSBs) of n-tier web applications. Using concrete experimental results we showed that VSI-DDoS attacks can be specially effective and stealthy because they can cause an intolerable long-tail latency issue of the target system while the average usage rate of all the system resources is at a moderate level (Sect. 2.3). We developed a VSI-DDoS attacking framework in which an attacker can systematically profile the target web application and train key attacking parameters for an effective VSI-DDoS attack (Sect. 3). Through a representative web application benchmark under realistic cloud scaling settings and equipped with the most popular state-of-the-art DDoS defense tools, we validated the negative impact and stealthiness of VSI-DDoS attacks, and confirmed the practicality of our attacking framework (Sect. 4). We further explored the pros and cons of two possible countermeasures for our attacks (Sect. 5). VSI-DDoS attack, as a newfound DDoS attack, is an important contribution to complement emerging DDoS attacks.

Acknowledgement. This research has been partially funded by National Science Foundation by CISE's CNS (1566443, 1566388), Louisiana Board of Regents under grant LEQSF (2015-18)-RD-A-11, and gifts, grants, or contracts from Fujitsu. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

References

1. Amazon Auto Scaling. https://aws.amazon.com/documentation/autoscaling
2. Amazon CloudWatch Concepts. http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html
3. ASA Threat Detection Functionality and Configuration. http://www.cisco.com/c/en/us/support/docs/security/asa-5500-x-series-next-generation-firewalls/113685-asa-threat-detection.html

4. Kaspersky DDoS Intelligence Report for Q1 2017. https://usa.kaspersky.com/about/press-releases/2017_kaspersky-lab-report-on-ddos-attacks-in-q1-2017-the-lull-before-the-storm
5. NSFCloud. https://www.cloudlab.us
6. RUBBoS. http://jmob.ow2.org/rubbos.html
7. Snort. https://www.snort.org/
8. Snort: The World's Most Widely Deployed IPS Technology. http://www.cisco.com/c/en/us/products/collateral/security/brief_c17-733286.html
9. Snort.AD. http://www.anomalydetection.info/?,32
10. Application DDoS Mitigation. Palo Alto Networks, Inc. (2014)
11. Clavister DoS and DDos Protection. Clavister, Inc. (2014)
12. Baset, S.A.: Cloud SLAs: present and future. ACM SIGOPS Oper. Syst. Rev. **46**(2), 57–66 (2012)
13. Curtis, K., Bodík, P., Armbrust, M., Fox, A., Franklin, M., Jordan, M., Patterson, D.: Determining SLO violations at compile time (2010)
14. Dean, J., Barroso, L.A.: The tail at scale. Commun. ACM **56**(2), 74–80 (2013)
15. Fayaz, S.K., Tobioka, Y., Sekar, V., Bailey, M.: Bohatei: flexible and elastic DDoS defense. In: USENIX Security (2015)
16. Gu, G., Zhang, J., Lee, W.: Botsniffer: detecting botnet command and control channels in network traffic. In: NDSS (2008)
17. Guirguis, M., Bestavros, A., Matta, I.: Exploiting the transients of adaptation for RoQ attacks on internet resources. In: IEEE ICNP (2004)
18. Herzberg, A., Shulman, H.: Socket overloading for fun and cache-poisoning. In: ACM ACSAC (2013)
19. IETF: RFC 6298. https://tools.ietf.org/search/rfc6298/
20. Jeon, M., He, Y., Kim, H., Elnikety, S., Rixner, S., Cox, A.L.: TPC: target-driven parallelism combining prediction and correction to reduce tail latency in interactive services. In: ACM ASPLOS (2016)
21. Jung, J., Krishnamurthy, B., Rabinovich, M.: Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In: WWW (2002)
22. Kang, M.S., Lee, S.B., Gligor, V.D.: The crossfire attack. In: IEEE S&P (2013)
23. Ke, Y.M., Chen, C.W., Hsiao, H.C., Perrig, A., Sekar, V.: CICADAS: congesting the internet with coordinated and decentralized pulsating attacks. In: AsiaCCS (2016)
24. Kohavi, R., Longbotham, R.: Online experiments: lessons learned. Computer **40**(9) (2007)
25. Kuzmanovic, A., Knightly, E.W.: Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In: ACM SIGCOMM (2003)
26. Li, J., Sharma, N.K., Ports, D.R., Gribble, S.D.: Tales of the tail: hardware, OS, and application-level sources of tail latency. In: ACM SoCC (2014)
27. Luo, X., Chang, R.K.: On a new class of pulsing denial-of-service attacks and the defense. In: NDSS (2005)
28. Mantas, G., Stakhanova, N., Gonzalez, H., Jazi, H.H., Ghorbani, A.A.: Application-layer denial of service attacks: taxonomy and survey. Int. J. Inf. Comput. Secur. **7**(2–4), 216–239 (2015)
29. Mathew, S.: Caching HTTP POST Requests&Responses. http://www.ebaytechblog.com/2012/08/20/caching-http-post-requests-and-responses
30. Mirkovic, J., Reiher, P.: A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Comput. Commun. Rev. **34**(2), 39–53 (2004)

31. Moore, D., Shannon, C., Brown, D.J., Voelker, G.M., Savage, S.: Inferring internet denial-of-service activity. In: USENIX Security (2001)
32. Oikonomou, G., Mirkovic, J.: Modeling human behavior for defense against flash-crowd attacks. In: IEEE ICC (2009)
33. Ramamurthy, P., Sekar, V., Akella, A., Krishnamurthy, B., Shaikh, A.: Remote profiling of resource constraints of web servers using mini-flash crowds. In: USENIX ATC (2008)
34. Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., Knightly, E.: DDos-shield: DDos-resilient scheduling to counter application layer attacks. IEEE/ACM Trans. Netw. (TON) **17**(1), 26–39 (2009)
35. Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Shimizu, T., Matsubara, M., Kawaba, M., Pu, C.: Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In: ICDCS (2013)
36. Wang, Q., Kanemasa, Y., Li, J., Lai, C., Cho, C., Nomura, Y., Pu, C.: Lightning in the cloud: a study of very short bottlenecks on n-tier web application performance. In: USENIX TRIOS (2014)
37. Xie, Y., Yu, S.Z.: Monitoring the application-layer DDoS attacks for popular websites. IEEE/ACM Trans. Netw. (TON) **17**(1), 15–25 (2009)
38. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Commun. Surv. Tutor. **15**(4), 2046–2069 (2013)
39. Zhang, Y., Mao, Z.M., Wang, J.: Low-rate TCP-targeted dos attack disrupts internet routing. In: NDSS (2007)