DAMUP: Practical and Privacy-aware Cloud-based DDoS Mitigation

Su-Chin Lin*, Po-Wei Huang*, Hsin-Yi Wang* and Hsu-Chun Hsiao*†
*Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
†Research Center for IT Innovation, Academia Sinica, Taiwan

***Abstract*—Cloud-based DDoS mitigation techniques have been widely deployed. However, existing approaches severely violate user privacy as they intercept HTTPS to uncover non-volumetric attacks. This paper presents DAMUP, DDoS Attack Mitigation Upholding Privacy. DAMUP is a practical and privacy-aware solution that empowers the cloud to blindly filter encrypted traffic while remaining deployable on the current Internet. The main idea underlying DAMUP is to encode the server's filtering policy, e.g., client's priority level or rate limit, into cryptographic tokens that can be stapled to an encrypted connection and be efficiently verified by the cloud. DAMUP is designed to be deployable on the current Internet because it requires no modification to the Internet architecture. For example, DAMUP leverages the Server Name Indication field in the TLS handshake to exchange tokens, and uses SOCKS proxy to override DNS. Our evaluation shows that DAMUP can significantly improve the connection success rate from 11.4% to 99.8% under HTTP(S) floods.**

I. Introduction

Distributed denial of service (DDoS) attacks are an escalating threat to web service availability due to their ease of weaponization and monetization. A DDoS attack can cost an enterprise at least $2.5 million [8], and increasingly more DDoS extortion scams thrive on web owners' fear of financial and reputation loss. Cloud-based DDoS mitigation has emerged as a mainstream technique that has been widely deployed in recent years because cloud (which lies between clients and servers) can naturally absorb attack traffic, remove attack traffic before it closes on the victim server, and require little modification on the server side. In cloud-based schemes, all traffic to the victim server is first redirected to the cloud (e.g., via DNS or BGP update) after which the attack traffic is filtered by the cloud based on its size and/or content.

Unfortunately, with the rise of low-rate (e.g., application-layer) DDoS attacks that leverage encrypted communication such as HTTPS [1], existing cloud-based schemes are either ineffective or they severely degrade user privacy. In particular, unlike volumetric DDoS attacks, low-rate DDoS attacks cannot be easily detected by simply monitoring the flow size, and thus the cloud needs to decrypt HTTPS for further inspection (e.g., by injecting a CAPTCHAs challenge to the client) [7]. However, the cloud acts as a man-in-the-middle that breaks the end-to-end security guarantee. An untrusted or compromised cloud could expose users' sensitive data (e.g., passwords and cookies) as demonstrated by the Cloudbleed vulnerability [2] against Cloudflare, one of the biggest DDoS defense providers.

To this end, an interesting research topic is to investigate whether cloud-based DDoS mitigation can guarantee user privacy while remaining effective and practically deployable. Herein, we propose DAMUP, a lightweight mechanism that enables private filtering by passing cryptographic tokens that encode the server's filtering policies. The main observation is that the victim server can often better differentiate benign and attack flows than the cloud because the server has additional knowledge about legitimate users (e.g., users who have registered before) and application-specific bottlenecks. In DAMUP, the server issues tokens to each authorized client and the client embeds tokens in flows that desire prioritized treatment in the case of DDoS. Consequently, the cloud can effectively block traffic based on the embedded tokens without inspecting the actual traffic content, thereby minimizing the trust on the cloud service. Prior work with similar observations has focused on empowering the server to control incoming packets [20], [15], whereas our work aims at preserving user privacy in a practical manner.

To be deployable on the Internet, the DAMUP solution must address several technical challenges. The first is how to efficiently construct tokens while preventing token misuse. DAMUP constructs the cryptographic token using a 256-bit MAC of the timestamps and client IDs (e.g., source IPs). In addition, the tokens are hierarchically derived to ensure efficient management without having to keep per client states. The second challenge is where to embed the token such that the cloud can retrieve the token without breaking end-to-end encryption.[1] DAMUP addresses this by embedding the token in the Server Name Indication (SNI) field supported by an TLS extension.[2] Finally, existing cloud-based defense can be circumvented when the server's IP is exposed [18]. To enforce DDoS filtering in DAMUP, the cloud tags valid packets, and only packets with tags can pass through the server's gateway. Because the number of protected servers (and thus their gateways) is much smaller than the number of clients, DAMUP efficiently implements such tagging by establishing an IPSec tunnel between the cloud and the server's gateway.

[1] One might attempt to embed the token in the unused TCP header fields but unused bits are considerably limited. Embedding the token in the HTTP header (e.g., adding a cookie) does not work because either the HTTP header is encrypted or it cannot be accessed by the cloud in our privacy-preserving setting.

[2] HTTPS means running HTTP over the Transport Layer Security (TLS) protocol. By design, the SNI field is left unencrypted to support multiple virtual hosts over HTTPS using the same IP and can be up to $2^{16}-1$ bytes.

978-1-5386-3416-5/18/$31.00 © 2018 IEEE

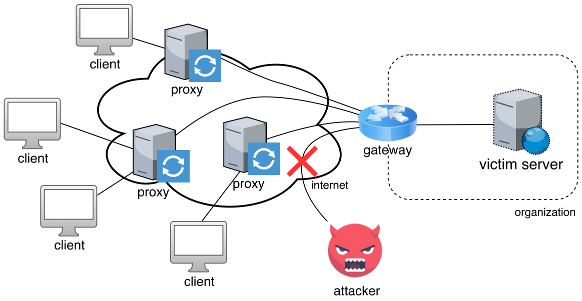

Fig. 1: Cloud-based DDoS protection.

In addition, DAMUP meticulously makes several design choices to minimize modifications on both client and server sides, as highlighted in Sections IV and V. Our evaluation shows that DAMUP can significantly improve the connection success rate from 11.4% to 99.8% under HTTP(S) floods.

II. Background and Related Work

Cloud-based DDoS protection involves three roles: *clients*, a victim *server*, and a *Cloud-based Security Provider (CBSP)*, which operates *proxies* on clouds. To enable DDoS protection, the victim server redirects traffic to the CBSP via DNS or BGP update. The CBSP then tries to identify and remove the attack traffic based on certain characteristics (e.g., size and content).

One vital advantage of cloud-based DDoS protection is deployability as it requires little modification to the Internet architecture and endpoints. However, existing CBSPs suffer from three critical limitations:

1) **Privacy violation**: A CBSP often requires access to the plaintext content in order to detect low-rate DDoS leveraging encrypted connections. For example, Radware decrypts HTTPS and injects a CAPTCHAs challenge to the client [7]. However, an untrusted or compromised CBSP could expose users' sensitive data [2].
2) **CBSP circumvention**: Because most CBSPs use DNS to redirect traffic, attackers can easily bypass the proxies if the victim's IP is exposed [18], [4].
3) **No destination control**: As pointed out by Liu et al. [15], CBSPs have full control over the scheduling and filtering decisions, and thus the destination cannot adopt per-destination, customized policies.

MiddlePolice [15] resolves the third limitation by enhancing cloud-based mitigation with network capabilities. However, MiddlePolice focuses on mitigating volumetric DDoS and requires cooperation with ISP to configure ACL. CDN-on-Demand [13] uses HTTP authentication cookie to identify privileged clients. For HTTPS websites, however, there may be a need to share the TLS key with the proxy, thus violating privacy.

Network-capability-based solutions, such as NetFence [14], do not meet the deployability requirements because they often require router upgrades and make it difficult for different ASes or ISPs to cooperate. Phalanx [11] uses an overlay to prevent DoS, which also requires router and ISP modifications. TVA [19] adds an additional field to the packet header, thus requiring router upgrades as well. Mirage [16] reaches per computation fairness by using crypto-puzzles to hide a victim's IP address. However, it needs IPv6 or upstream ISP support.

In an attempt to lessen privacy violation, Cloudflare supports a keyless SSL option [6], which allows the server to keep its long-term private key (instead of sharing it with the cloud); however, the server needs to maintain an on-demand decryption oracle to return the plaintext to the cloud. This approach remains unsatisfactory as the cloud can still access users' sensitive data. Therefore, this work focuses on addressing the first and second limitations. Since our scheme is inspired by network capabilities, it naturally supports destination control.

III. Problem Definition

A. System Model

Network and bottleneck. As shown in Figure 1, an end-to-end connection can be divided into three segments, which are client-proxy, proxy-gateway, and gateway-server. Similar to other cloud-based approaches, we focus on bottlenecks residing on the gateway-server or the server itself. Proxies usually have a relatively large network link capacity and computational resources, as they can be hosted on multiple data centers; gateways are edge routers of an organization, and most enterprise gateways have gigabit-level network link capacity. Thus, the server's capacity is bounded by its computational efficacy and the gateway-server link capacity.

Server and client. Clients can be benign or malicious. We assume the victim server[3] is able to differentiate whether a client is benign or not when the client makes its initial connection, and establish a shared secret with the benign one. For example, the server may require login and ask the client to provide identity proof during registration, such as a verifiable email or ID. The client is regarded as benign to the server after successful registration and verification. Note that the initial connection can be protected using existing heavyweight schemes [17], and we focus on providing lightweight DDoS protection for the subsequent connections.

B. Threat Model

The goal of the attacker is to disrupt the service using DDoS. The attacker controls a large number of bots that can send traffic to flood the gateway-server link or paralyze the server. We focus on attacks that target the gateway-server link or the server because a previous study [3] showed that 80% of DDoS attacks sent 1 Gbps traffic and only 1% of them sent 10 Gbps traffic. In other words, even if the attacker can successfully bypass the cloud, only a small number of attackers would be capable of DDoS attacks on the gateway directly.

We consider flash crowds (i.e., benign human users flooding the server) outside the scope of this paper because the cost of controlling a large number of human users with verified identities is too high for the attacker.

[3] More precisely, we would like to protect the service or application on the victim server. For ease of description, we will simply use "server" throughout this paper.

C. Desired Properties

To overcome the limitations stated in Section II while remaining deployable, a cloud-based DDoS mitigation scheme should satisfy the following desired properties.

Filter traffic while preserving privacy and transparency. Usually, the client traffic is filtered at the proxy. Traditional solutions require the inspection of packet contents, and for HTTPs packets, the proxy requires the server's private key. Several CDN hijacked events have shown that it is not secure to put private keys on proxies. If the proxy is compromised, sharing private keys would cause a leakage of packet contents. Besides privacy issues, filtering packets on proxies means that the server cannot decide which kind of packets should be filtered, since the total control is held by the proxy. A cloud-based DDoS mitigation scheme should support filtering polices without requiring the server's private keys or the ability to analyze packet payloads and thus let the server control the filtering process to a greater extent.

Protect public address servers. Cloud-based DDoS protection often requires hiding server's origin IP address. However, once the IP address is leaked, the attacker can easily bypass the proxy and connect to the server directly. Besides, it is inconvenient for some organizations to change all their servers' IPs to private. Hence, a cloud-based DDoS mitigation scheme should be able to protect servers with public IP addresses. Even if clients intend to bypass the proxy, the attack traffic should be intercepted before it reaches the server.

Affordable and feasible. A feasible protection should be deployable on the current Internet. It would be ideal to utilize upstream ASes such as ISPs, but such an option may be impractical sometimes. Some network equipment supports advanced functionality for DDoS protection, but most of such equipment is expensive or difficult to deploy. A cloud-based DDoS mitigation scheme should be deployable on common network systems with minimal modification and requirements demand on the original structures, including the server and client sides.

IV. DAMUP Design

We propose a token-based DDoS defense mechanism called DAMUP. At a high level, DAMUP utilizes a shared secret (between a benign client and server) established during the initial connection to derive tokens. The client marks its traffic as benign by embedding tokens. Every connection to the server is redirected through the DAMUP proxy, and those without valid tokens will be blocked by the proxy. The benign ones (with valid tokens) are then passed to the server via a secure tunnel, further protecting the server from attacks that somehow bypass the proxy. As a results, the DAMUP proxy successfully filter malicious traffic while remaining oblivious about the traffic content.

This section describes the major components in DAMUP, the step-by-step protocol flow, and a hierarchical key derivation method to reduce token management cost.

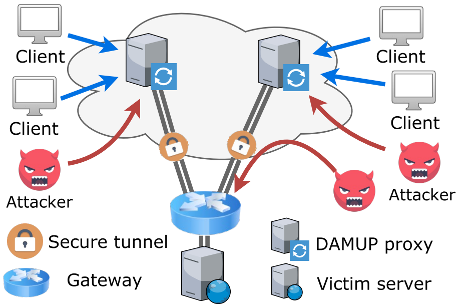

Fig. 2: DAMUP system architecture

A. Major Components

DAMUP contains four major components — **token manager**, **proxy**, **gateway**, and **server**, as shown in Figure 2.

Token manager. To simplify the token management task on the client side, each benign client connected to the DAMUP installs a lightweight token manager program, which stores the client's secret key in advance and automatically derives tokens based on the secret key. Whenever the client makes a request to a DAMUP-protected server, the token manager will embed the corresponding token into the request.

The DAMUP proxy. Deployed in clouds, proxies can usually resist higher intensity attacks as explained in Section II. Each DAMUP proxy will establish a secure tunnel to the server's gateway. When the server is not under attack, the DAMUP proxy allows every packet pass through the secure tunnel. In contrast, when the server is under attack, the DAMUP proxy will block connections without valid tokens and let those with legal tokens to pass through the secure tunnel. Additionally, the DAMUP proxy monitors the sending rate associated with each token to prevent overuse. Once the rate of a token exceeds a certain threshold, all subsequent traffic associated with the token will by dropped by the DAMUP proxy. A client's secret key will be revoked if the client overuses its tokens multiple times. Thereby, the DAMUP solution ensures content privacy since the DAMUP proxy does not have to know the server's TLS private key.

The Gateway. To reduce the damage when the server's public address is exposed, DAMUP constructs a secure tunnel between each DAMUP proxy and the gateway. The gateway router will only forward packets that come from the secure tunnel to the server. Hence, if the attacker attempts to directly access the server via the server's IP address, even if the attacker can pretend to be a DAMUP proxy via IP spoofing, the packets will be dropped by the gateway.

The server. There are only two modifications on the server. First, it needs to redirect all traffic to our DAMUP proxy in cloud via a DNS override. Second, a new web page is needed for identity verification and key distribution. The server can also maintain a whitelist of clients who are authorized to get valid tokens. Hence, DAMUP also enhances transparency of filtering policies because it allows a server to define per-client policies, whereas most commercial cloud-based solutions consider filtering policies as their business secret.

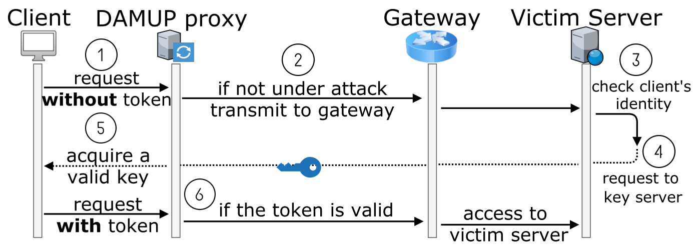

Fig. 3: DAMUP protocol flow

B. Protocol Flow

Figure 3 describes each step of the process by which a registered client obtains a valid key from the server.

1) When a new client makes the initial request to a DAMUP-protected server, the client will be redirected to the DAMUP proxy in cloud. At this time, the client has no shared secret with the server or token manager.
2) If the server is under attack, the DAMUP proxy will block any connection without a valid token. When the server is not under attack, the DAMUP proxy will allow every connection to pass, and therefore this client can connect to the server.
3) During the client's first access to the server, the client will be verified and a client-side token manager will be installed to manage the secret key and tokens.
4) If the client's identity is checked out, the server will return a shared key to the client. The keying service can be maintained by the server or delegated to the proxy.
5) The token manager stores the client's key in local storage.
6) When the client makes a subsequent request, the token manager derives a token using the secret key and embed it into the request, such that the client can pass thorough the DAMUP proxy and reach the server.

DAMUP uses a long enough cryptographic token (e.g., 256- bit) to ensure security. To prevent attackers from abusing tokens, each token has a specific lifespan T and a limited throughput. When a token expires, the token manager will automatically calculate the next token using the secret key. Even if the attacker can somehow guess or steal a valid token, the attacker can send at most a threshold amount of attack traffic before the token expires.

C. Hierarchical Key and Token Management

DAMUP maintains keys in a hierarchical manner (Figure 4) to reduce key management cost. DAMUP assigns the server a long-term key K_s which should be appropriately protected. For simplicity, we assume there is only one DAMUP proxy in cloud. With this long-term key K_s, the server can derive a series of short-term proxy keys K_{p_i} for the proxy based on a key derivative function (KDF) such as HMAC(key = K_s, msg = timestamp). The timestamp described above is a UNIX timestamp divided by the refresh time interval (*e.g.,* one month). Therefore even if the proxy is compromised, the compromised key will be invalidated after one refresh interval. As shown in the lower part of Figure 4, the proxy with the proxy key can derive a series of secret keys for authorized clients based on KDF(K_{p_i}, client_id). With this secret key, the token manager of an authorized client can generate valid tokens and automatically pass through the DAMUP proxies' filtering mechanism. To validate a token, the DAMUP proxy simply checks whether the token has come from the secret key derived by KDF(K_{p_i}, client_id).

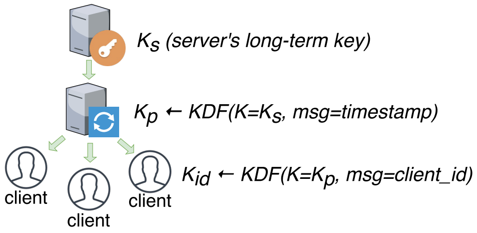
Ks (server's long-term key)
Kp ← KDF(K=Ks, msg=timestamp)
client
client
client
Kid ← KDF(K=Kp, msg=client_id)

Fig. 4: Hierarchical key and token management

The attacker might steal the client secret keys and generate numerous valid tokens to attack the server. Hence, DAMUP should be able to revoke a client's secret key. The DAMUP proxy will record the revoked secret key in a Revoke Key List.

V. DAMUP Implementation

A. Token Derivation and Validation

1) Server Name Indication: To filter TLS connections without breaking end-to-end encryption, DAMUP embeds the token in the Server Name Indication (SNI) [9] field in the TLS handshake protocol. SNI is a TLS extension that is widely implemented and supported in modern browsers nowadays. The extension is used to indicate the hostname the client connects to and thus allows the server with multiple TLS certificates to present the corresponding certificate when processing the incoming TLS handshake. Specifically, the SNI will be included in plaintext in the Client Hello message (the first handshake message), such that the server can identify the hostname before the encrypted session starts.

To modify SNI, we override the hostname in the URL address. Neither root permission nor intercepting a HTTPS request is required. This approach is particularly advantageous since HTTPS interception leads to privacy concerns [12]. In addition, because the token will be placed in the URL, the user can be aware that the protection is active.

2) Token Validation: To validate the token, the DAMUP proxy first inspects the SNI field in TLS extension. If the token in SNI is correct, it simply proxies the TCP connections to the server. Otherwise, the connection is either aborted or falls back to an error page if the SNI is invalid or empty. The default behavior of handling invalid tokens can be determined by the server. We briefly explain the pros and cons.

We can simply abort the connection, but new clients without tokens may be affected. These users may falsely infer that the website is down. By contrast, a fallback to an error page provides the new client useful information. However, in the error page, the TLS certificate will be invalid, since the DAMUP proxy does not have the TLS private key of the server.

B. DAMUP *Proxy*

1) Secure Tunnel: The DAMUP proxy establishes a site-to-site VPN tunnel with the gateway, such that the server is able to tunnel traffic through this secure channel. We utilize IPsec to build an IKEv2 VPN with pre-shared keys. IPsec provides data origin authentication and thus the tunnel is resistant to IP address spoofing.

The tunnel serves two purposes. First, it ensures that all traffic from the user can be tunneled to the server via the DAMUP proxy. This is the basic cloud-based DDoS mitigation architecture. Second, all token derivation and negotiation process is transmitted via the tunnel. IPsec provides data confidentiality, which means all of the traffic is encrypted.

2) Cloud-based: As described in the threat model, the network bandwidth and computational resources of the DAMUP proxy should be as large as possible. Hence, cloud services can be utilized to deploy the DAMUP proxy. These services not only meet the demands of the proxy but are also affordable. Moreover, DAMUP proxies can be geographically distributed to enhance the ability to disperse and absorb attack traffic.

C. Server Implementation Details

1) Change DNS record: If the DNS name server of the website can be changed, we can simply change the `NS` record to the customized name server of the DAMUP proxy. A wildcard DNS record will be set up, pointing to the DAMUP proxy. Nevertheless, some websites may be unable to change the DNS name server. Hence, we provide another option: change the `CNAME` record. In this option, the `A` record of the server will be replaced by a `CNAME` entry. The `CNAME` record points the hostname to the DAMUP proxy. Therefore, it can proxy the client traffic to the server.

2) Wildcard TLS Certificates: Because we leverage the domain name to modify SNI, a wildcard TLS certificate is required to avoid triggering browser warnings. That is, the `Common Name` in the certificate has to contain the specific domain name prefixed by the tokens. Otherwise, the browser will warn the user due to mismatch error. Therefore, a wildcard certificate (e.g., `*.example.com`) is required. Note that using a multi-domain certificate is infeasible because the number of possible tokens is large (thus the subdomains). It is worth mentioning that wildcard certificates are becoming more and more ubiquitous. The world's largest free certificate authority, Let's Encrypt, plans to offer wildcard certificates in February 2018 [5].

3) DNS Override: A DAMUP client modifies SNI by locally updating the domain name of a connection request. However, because the domain name prepended with a valid token does not exist in any DNS records, the DNS has to be overridden. We choose SOCKS V5 proxy as a solution because it is capable of performing remote DNS lookups. Additionally, most modern browsers support SOCKS5 and the Proxy Auto-Configuration (PAC) file. The PAC file defines the hostname of the proxy server, and whether the browser proxies the request for a given domain name. Given below is a PAC file example. The SOCKS proxy server is simply the same server of the DAMUP proxy.

```
if (dnsDomainIs(host, ”example.com”))
    return ”SOCKS damup-proxy.com:12345”;
```

In the code above, only the requests under the subdomain of `example.com` will proxy through the DAMUP proxy. This ensures a user's privacy and security.

If the DNS name server record can be changed, the user does not need to use a SOCKS proxy because the domain name exists in the DNS records.

4) Token manager: In order to differentiate benign flows from the malicious ones, a DAMUP proxy requires its clients to place tokens in the SNI field. We implement a cross-browser extension based on the WebExtensions API. The browser extension will store the client's secret key in the local storage in advance. Then, when the client makes a request to the target domain, the browser extension will intercept it and place the secret token in the SNI header.

Specifically, if an authorized client makes an HTTPS request to `example.com`, this extension will intercept this request before any TCP connection, and then calculate the secret-token using HMAC of the *client secret key* and *UNIX timestamp / refresh interval* T. The extension will modify the URL of this request to `secret-token.client_id.example.com`. As mentioned in Section IV-C, the token will be updated after every T seconds to ensure security.

VI. Evaluation

Testbed. To simulate a server with limited computational resources and network throughput, a Raspberry Pi 3 running `NGINX` HTTP server is set up as the server. The server is connected with a router through a 100 Mbps Ethernet interface. Both uplinks of the router and proxy are 1 Gbps.

Since all cloud-based schemes have path detour latency, we focus on evaluating the overhead introduced by DAMUP's computational and communication overhead. Hence, in our experiments, both the router and the DAMUP proxy have public IP addresses under the same subnet, which is intended to minimize the detour latency. In addition, an IPsec VPN tunnel is established between the two public IP addresses.

Latency. The latency evaluation compares DAMUP with a `vanilla` scenario where the naked server is configured without any proxy server. In name, a DAMUP proxy is set up to proxy users requests. The DNS `A` record of the server is replaced with a `CNAME` record pointing to the hostname of the proxy. Additionally, the user already has a valid key, and utilizes the SOCKS5 proxy for remote DNS lookups. We compare the latency of downloading 1 MB, 500 KB, and 100 KB PNG images respectively from the server. Each experiment is repeated 1000 times.

The results are shown in Figure 5. When downloading 500 KB and 1 MB images, the extra latency is almost negligible. After the TCP connection is established, the cost of transmitting the file is low. However, when downloading 100 KB images, the extra latency becomes larger because when using the DAMUP solution, the user has to send the request via the DAMUP proxy. The overhead mainly comprises two parts: the

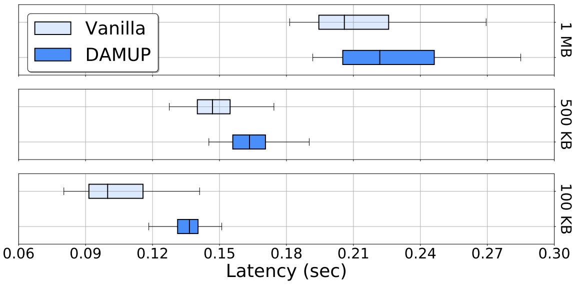

Fig. 5: The test result of latency

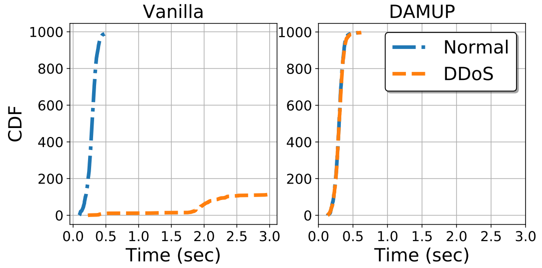

Fig. 6: The test result of mitigation

query of remote DNS through SOCKS proxy, and the extra overhead of proxying the HTTP request. Of the 36.66% extra overhead (compared with the `vanilla` request), 33.23% is due to the SOCKS proxy. If we are able to override the user's DNS request locally, the overhead can be reduced substantially since it avoids sending request via the SOCKS proxy. However, since overriding DNS locally requires user interaction, we anticipate that using the SOCKS proxy is still a highly usable, deployable solution.

Mitigation. We deployed 16 computers as malicious bots. In the `vanilla` (naked server) scenario, the bots simply perform a HTTPS GET DoS attack to overwhelm the server; the benign users are simulated from a single machine. 6 users will download a 500 KB image simultaneously, with a total of 1000 requests. Figure 6 shows the evaluation results.

In the `vanilla` scenario, each bot overwhelms the naked server by flooding HTTPS GET requests. For the benign user, the request is considered failed if the response time is more than 3 seconds. Figure 6 shows that 886 requests fail to download the image, and only 11.4% of the requests are successful. For those successful connections, the latency increases dramatically. Compared to the normal case, the mean of latency is approximately 6.6 times larger. In the DAMUP scenario, each bot uses a fake token to attack the server. Our result shows that very few requests failed — only 0.2%. Moreover, the difference of latency is almost negligible. To summarize, the cloud-based architecture of DAMUP is effective to mitigate DDoS attacks.

VII. Conclusion

In this paper, we propose a privacy-preserving, practical, and transparent DDoS mitigation architecture. DAMUP highlights three contributions. First and foremost, DAMUP focuses on the privacy. According to existing research [10], 76.5% of organizations share the private key with a third-party provider; consequently, user privacy is severely violated. Our proposed methodology not only preserves privacy, but also performs effective DDoS mitigation. Secondly, DAMUP is highly practical, deployable and scalable. While related works require the cooperation of ISPs, ASes, DAMUP focuses on practicality. Without modifying network infrastructure and much user interaction, we utilize SNI to implement the secure tokens. Finally yet importantly, unlike typical enterprise cloud-based approaches, we adopt a transparent policy to mitigate DDoS attacks. The server is able to define who the benign users are.

Acknowledgments

This work was supported in part by Taiwan Information Security Center (TWISC), Academia Sinica, and the Ministry of Science and Technology of Taiwan under grants MOST 105-2221-E-002-146-MY2, 106-3114-E-002-005 and 107-2636-E-002-005-.

References

[1] "Application-layer DDoS Attacks: The Numbers May Surprise You," https://www.arbornetworks.com/blog/insight/application-layer-ddos-attacks-the-numbers-may-surprise-you.

[2] "Cloudbleed bug: Everything you need to know," https://www.cnet.com/how-to/cloudbleed-bug-everything-you-need-to-know.

[3] "DDoS Attacks Size," https://www.helpnetsecurity.com/2016/07/19/ddos-attacks-escalate/.

[4] "HatCloud - Bypass CloudFlare with Ruby," https://github.com/HatBashBR/HatCloud.

[5] "Looking forward to 2018," https://letsencrypt.org/2017/12/07/looking-forward-to-2018.html, accessed: 2018-01-23.

[6] "Overview of Keyless SSL," https://www.cloudflare.com/ssl/keyless-ssl/.

[7] "Radware's ssl-tls attack mitigation solution," https://www.radware.com/solutions/ssl-attack-protection.

[8] "The Average DDoS Attack Cost for Businesses Rises to Over 2.5m," http://www.zdnet.com/article/the-average-ddos-attack-cost-for-businesses-rises-to-over-2-5m.

[9] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport layer security (tls) extensions," Internet Requests for Comments, RFC Editor, RFC 3546, June 2003.

[10] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Measurement and analysis of private key sharing in the https ecosystem," in *ACM CCS*, 2016.

[11] T. A. Colin Dixon, "Phalanx: Withstanding Multimillion-Node Botnets," in *USENIX NSDI*, 2008.

[12] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception," in *NDSS*, 2017.

[13] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "Cdn-on-demand: An affordable ddos defense via untrusted clouds," in *NDSS*, 2016.

[14] X. Liu, X. Yang, and Y. Xia, "Netfence: preventing internet denial of service from inside out," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 255–266, 2010.

[15] Z. Liu, H. Jin, Y.-c. Hu, and M. Bailey, "MiddlePolice: Toward Enforcing Destination-Defined Policies in the Middle of the Internet," in *ACM CCS*, 2016.

[16] P. Mittal, D. Kim, Y.-C. Hu, and M. Caesar, "Mirage: Towards deployable ddos defense for web applications," *arXiv preprint arXiv:1110.1060*, 2011.

[17] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks," in *Proceedings of ACM SIGCOMM*, 2007.

[18] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis, "Maneuvering Around Clouds: Bypassing Cloud-based Security Providers," in *ACM CCS*, 2015.

[19] D. W. Xiaowei Yang and T. Anderson, "A DoS-limiting Network Architecture," *ACM SIGCOMM CCR*, 2005.

[20] A. Yaar, A. Perrig, and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *IEEE S&P*, 2004.