Revisiting Remote-Attack-Prevention: Challenges in Achieving Complete Decentralization

Abhinav Aggarwal
University of New Mexico
Albuquerque, NM
abhiag@unm.edu

Mahdi Zamani
Visa Research
Palo Alto, CA
mzamani@visa.com

Mihai Christodorescu
Visa Research
Palo Alto, CA
mchristo@visa.com

ABSTRACT

In this paper, we revisit the notion of remote-attack-packet-filtering under the umbrella of *global software-defined networking*, in which a user on the Internet is allowed to remotely configure a security gateway through an interactive and verifiable traffic policy learning on top of careful service-level negotiations. This would enable the victim of an attack to (provably) block the attack packets closer to their origin point. While many solutions have been proposed for this problem, all existing work relies on some notion of trust between the entity requesting protection and the entity providing this service. Since trust is rare on the Internet, we seek a fully autonomous and decentralized solution. This problem can be deceptively challenging and poses some interesting and novel technical roadblocks of its own. We highlight some of these challenges in this paper and layout some open problems.

As a case study, we consider two past global-scale attacks, Mirai and Wannacry, and use their reported scale and impact on particular organizations to estimate the feasibility of an incentive-driven approach for the problem, where the victim negotiates and pays a reward to the remote entity (middleboxes) for its protection. Our analysis of Mirai estimates that negotiated protection from only 10 middleboxes would be sufficient to cut the attack volume by 40%, and that the financial incentive can be ramped up to $150,000 per middlebox while preserving the victim's cost effectiveness.

1 INTRODUCTION

Enterprise and home networks are connected to the Internet via security gateways such as firewalls, intrusion-detection systems, and data leakage-protection systems. These gateways are designed to protect the local network against attacks originating from the Internet. Such a network architecture is widely prevalent, to the point that most network routers have built-in security functions. While these gateways often protect the local network from malicious incoming traffic, they usually are not deployed to protect the Internet from malicious outgoing traffic.

Some security gateways can filter outgoing traffic, typically to protect the local network's confidentiality by preventing the ex-filtration of sensitive data. For example, such filtering applies to outgoing traffic destined to or originated from certain ports (e.g., blocking known malware-related ports or broadcast traffic) or with certain contents (e.g., blocking HTTP requests to known-bad servers). The network administrator can manage such outgoing filtering to improve the security of the local network. Unfortunately, in most cases, the security of the rest of the Internet is not considered, leading to frequent and widespread infection of networks with malware that are not mitigated at much smaller scale in local networks.

In this paper, we propose to add another purpose to security gateways, by tasking them with protecting the Internet at large from attacks originating from the local network. We call this approach *global software-defined networking (*global-*SDN)* to emphasize the use of software-run gateways that can help prevent attacks not only on a local level but also on a global scale. Our goal is to enable a server to automatically configure a gateway across the Internet to block malicious traffic that reaches the server after passing through the gateway. The server can compensate the gateway for its service to incentivize participation in the protocol.

How can remote-filtering help? Filtering malicious packets at a location other than the attack target may improve the efficiency of attack detection for three reasons: (1) *Real-time detection of malicious packets* is a needle-in-a-haystack problem for most security gateways, especially among a large volume of incoming traffic [18]. A gateway likely receives smaller amounts of traffic than a busy web server – its traffic is likely easier to classify, and thus, detecting the malicious packets before they are combined with a large amount of benign traffic can potentially be done with significantly better accuracy; (2) Malware can infect a large number of nodes in a matter of minutes to a few hours [15], accelerated by today's always-connected Internet nodes. Therefore, detecting and filtering malicious packets from a point close to their origin could *slow down malware propagation*; (3) There are often multiple security gateways on the way between a host and a web server [14]. Depending on the type of the attack, the server may *choose to configure a* *gateway* that is located closer to the attack source to achieve a more accurate policy. For example, in response to a DoS attack, the server may choose to install the policy on a closer gateway, where the attack traffic is aggregated. On the other hand, if the server detects a centralized DoS, then it may choose a farther gateway to reduce the false-positive rate as such a gateway would likely receive less traffic than a closer gateway.

Why should a server outsource its network security policy to a security gateway not in its control? The main goal for the server is to improve its network security, both in terms of accuracy and cost. In terms of accuracy, remotely configuring the policy of a security gateway allows the server to achieve a security policy that is customized to the attacker's network. This is because the security policy resulting from the remote-configuration process takes into account both the attack traffic that is afflicting the server and the normal traffic that passes through the security gateway. As a simple example, denial-of-service attack traffic is best blocked by filtering it as close to the attacker as possible, as filtering it later in the network becomes almost impossible due to bandwidth constraints and due to similarity to server's normal incoming traffic.

In terms of cost, remote policy configuration effectively offloads some of the security enforcement typically done at the server's own firewall to the security gateway. This is particularly useful for servers who run popular Internet-facing services and would need extremely powerful firewalls to handle all of the incoming traffic at a single ingress point. As a matter of fact, highly scalable Internet services often operate in a distributed fashion from multiple data centers, complicating the security management of all the firewalls involved. Offloading the security policy to remote security gateways would allow for cheaper and easier management of server-local firewalls, while making use of the large number of security gateways sitting mostly idle in front of the users accessing those servers.

Why should a security gateway allow an untrusted party change its security policy? The security gateway operates under one requirement, to allow all desired traffic sent by the network it protects to reach the Internet. In general, the definition of *desired traffic* is not known, not even to the security gateway, and thus the gateway mostly operates by allowing traffic that it has seen before or that was explicitly whitelisted. In this context, the request to change the security policy sent by the server (which is an arbitrary node on the Internet from the point of view of the security gateway) introduces two problems. First, the gateway has to account for the additional resources to be spent configuring and deploying a new security policy. Second, the gateway

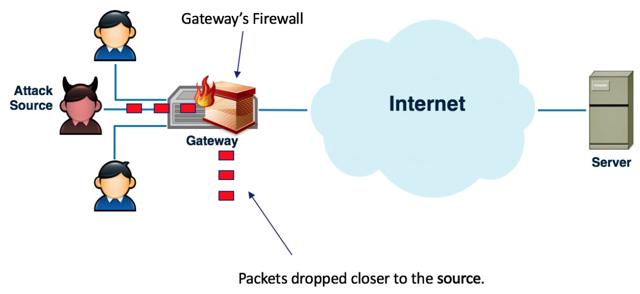

Figure 1: Global-SDN model

has to ensure that the new policy still allows all desired traffic to reach the Internet.

To motivate the security gateway, we propose that it is paid for the resources it spends (e.g., additional computation time, additional memory, additional network latency) to handle the security policy requested by the server. To ensure that the new policy still satisfies the existing needs of the security gateway's protected network, the existing security policies at the gateway provide a good starting point. This, however, poses some non-trivial security challenges, which we highlight in the later sections of this paper.

Isn't *global*-SDN just a security-specific application of the long-dead Active Networks framework? Our proposal could be realized on top of an Active Network [25], where the server could provision an active application on a gateway in order to enforce the desired security policy. We see two current trends that make the *global*-SDN framework significantly more practical than Active Networks. Programmable network processors, also known as Smart-NICs [29], are widely available commercially and are becoming part of router and middlebox designs, thus providing programmable packet processing at high speeds. Machine learning has made significant progress in the last decade, opening the door to creating traffic policies from examples, removing the need for routers and administrators to agree on a common programming environment as in Active Networks.

An overview of the challenges. Depending on how the server and the gateway are allowed to behave, designing a *global*-SDN protocol can become a challenging problem. In a realistic model, both parties can play *rationally* maximizing their payoffs. For example, a server can modify a gateway's policy accidentally or maliciously to impact other servers' traffic. Also, the gateway can collude with malicious clients to deploy a malicious policy causing delays in attack prevention; or, the gateway can take advantage of the *global*-SDN protocol to ransom the server demanding money to stop the attacks.

A possible strategy is for the server to help the gateway *learn* a traffic policy through interactive exchange of information regarding what traffic needs to be monitored or blocked.

Once a sufficiently accurate policy is learned, the gateway can deploy configure its firewall to include this policy. Such an interaction may also include negotiation on a *service fee* that the server will pay to incentivize gateway's honest participation in the protocol. Figure 1 shows the key participants in such a system.

An alternate strategy for the server is to create the policy on its own and send it to gateway for deployment only. This is not desirable because (1) the gateway would need to check that the server-created policy is not malicious, a hard problem; (2) the gateway and the server would need to agree on a common policy language; and (3) a server-created policy may restrict the gateway's opportunities for optimizing policies across multiple, unrelated servers.

2 RELATED WORK

There are various existing examples of security gateways whose policies are remotely configured. While a significant fraction of these examples take the usual approach of policy configuration by the actual administrator of the security gateway (even when this administrator performs the policy configuration via remote access), other mechanisms that (slightly) relax such a privileged access exist. In particular, techniques like configuring the Network Address Translation (NAT) capability of a firewall lead to the creation of Internet Gateway Device (IGD) protocol [3] part of the Universal Plug and Play (UPnP) suite, the NAT Protocol Mapping Protocol (NAT-PMP) [4], and the Port Control Protocol (PCP) [31]. All of these protocols are focused on configuring NAT gateways and NAT firewalls and, as a result, are not a general solution for security gateways that go beyond simple firewall-like security policies. However, they assume a level of trust between the security gateway and the policy-requesting device. We do not consider such protocols to be good candidates for global SDN and make no assumption of trust between the server and the gateway in our model.

The need to deploy traffic policies closer to the source is illustrated by two state-of-the-art systems. Operators of Content Delivery Networks (CDNs) often employ their network of nodes to drop or otherwise limit network traffic that is undesired by their customers. In particular, the case of Denial of Service (DoS) attacks shows that a moving server nodes closer to the source of a request not only provides geographic redundancy but only allows for the fast deployment of traffic policies to block DoS packets. Google's Project Shield [1] and related patents [6, 19] achieve various forms of *remote network security management*, to provide security protection via traffic policies enforced at a Remote Management Center. We note that both of these examples are fundamentally centralized, as the security policy is created, deployed, monitored, and updated by one party that is trusted by the servers it protects and that controls the gateways enforcing the policy.

3 WHAT MUST A SOLUTION TO *GLOBAL*-SDN LOOK LIKE?

To design a robust and scalable *global*-SDN system, several challenges need to be addressed. Underlying the goals of improved accuracy and lowered cost is the key need to maintain the same security guarantees as in the local firewall scenario. Because the enforcement of the security policy is now shifted to a remote security gateway not controlled by the server, the system needs to explicitly account for the potential loss of trust. Moreover, moving the security policy onto a local network's gateway presents new challenges related to computational requirements and to deployment. If, for example, Amazon decides to offload its security policy from its security firewall to the security firewalls of its top-100 million customers, then each of those customers will presumably have to prepare their firewalls to handle the increased traffic filtering task. Furthermore, if only some small percentage of those top-100 million customers are willing to participate in the system, Amazon may decide that the effort of offloading only a small part of their security firewall is not worth the small benefit to be gained. Thus, one has to design incentives for both parties to participate, in addition to addressing the newly introduced security problems.

System model. As mentioned before, we only focus on three key players in this paper: the *server*, which is the victim of the attack; the *adversary*, which is sending attack packets to the server; and the *gateway*, whose policy the server wants to remotely configure to impede the attack traffic targeted at itself by the adversarially-controlled clients.

We allow the adversary to communicate to the Internet through one or more connection points, and generate any type, volume, and patterns of traffic it wishes. Furthermore, we do not assume that the gateways involved in the *global*-SDN architecture are trusted prior to the beginning of the algorithm.

We do not make assumptions about how an arbitrary gateway on the Internet will react to the protocol, as long as it is not directly controlled by the adversary (we do allow possibility of collusion though). Gateways could be (1) honest; or (2) rationally benign (trying to maximize profits without launching attacks and without colluding with the adversary); or (3) rationally malicious (maximizing profits by launching attacks or colluding with the adversary to launch attacks); or (4) irrational (profits are not important). For the sake of abstraction and keeping the discussion at a relatively high level, we will use the terms *middlebox* and *security gateway* interchangeably in the remainder of the paper.

Finally, a realistic assumption about the server is for it to be rational, in that the server wishes to get the best protection at minimum cost. It can take advantage of the *global*-SDN protocol to deploy a policy that benefits the server, but not to the rest of the Internet. For example, the server may attempt to deploy a policy that blocks traffic to other servers.

The reader is again referred to Figure 1 for a clearer picture of the main players in the system. We only deal with the scenario of a single server (victim) and single adversary in this paper. The extension to a more general setting is currently much beyond the scope of this project.

3.1 Security Challenges

Under the setting just described, a primary security challenge that any solution to *global*-SDN must address is for the server to (autonomously) be able to (1) find a gateway to run the protocol with; (2) verify that the gateway is indeed involved in routing the attack packets to the server; and (3) run a mechanism to trust the gateway enough to let it control the incoming traffic to the server. We refer to this problem as the ***Gateway Discovery Problem***. In what follows, we further discuss this problem in detail and highlight the various challenges it poses.

3.1.1 Discovery Mechanism. How does the server find a gateway with which it can run the *global*-SDN protocol and which it can trust enough with controlling its incoming traffic? Since our vision is to not rely on any central (trusted) authority or mechanism for remote access control, this search must be done under minimal trust assumptions on the middleboxes in the Internet.

The challenge in enumerating the gateways on the path between two Internet nodes lies in the fact that there may be multiple paths between such nodes, each one used at different times or for different types of messages or for different directions of communication. The Internet is designed as a potentially directed communication graph, where routing policies can move traffic along unexpected routes given the observable topology. Commercial agreements driven by cost and traffic volumes, load balancing, privacy requirements, regulatory constraints, and availability and reliability needs ensure that routing policies at all levels (from local networks to large autonomous systems) are complex, often hard to infer, and uncertain to rely upon.

While it is reasonable to assume that on most paths, the average distance between different autonomous systems is sufficiently large [14], it is also encouraging to note that the scenario in which the adversary takes over all these intermediate links and launchs a full scale directed attack at a particular server is improbable in practice. Of course, in case it happens, the only option for the server would be to configure its own firewall. For a more general situation though, one could potentially build protocols on top of Traceroute [16, 23] to determine this list of gateways. However, care must be taken to not broadcast too many messages during this tracing phase to prevent the server from launching a DoS against itself or the neighboring nodes. Moreover, using Traceroute directly might not reveal all the information about the gateways on the path from the server and the attack source, either due to the security settings of these gateways or various other reasons. To the best of our knowledge, the requirements of *global*-SDN are strictly stronger than what current implementations of such tools can provide.

3.1.2 Authentication of Communication. How do the server and the gateway communicate to each other in a way that prevents an adversary from (among many other things) (1) eavesdropping the conversation and controlling the flow and contents of the attack packets in a way that invalidates the filter being learnt at the gateway; and (2) running a man-in-the-middle attack in order to obtain the reward offered by the server in exchange for the filtering service? In an ideal scenario when the gateway is an arbitrary but a sufficiently enriched middlebox on the Internet, it becomes important for the server and the gateway to establish an authenticated and secure connection over which the *global*-SDN protocol will run, or else, the adversary can completely compromise the security of the underlying task. Establishing such a secure connection can be expensive, nonetheless, an essential requirement here, specially when the aim is to derive trust within a completely-anonymous network.

3.1.3 Proof of Routing. In order to stop a (sufficiently) large fraction of incoming attack packets and balance the costs involved in interacting with remote gateways, the server needs to verify if the attack packets are actually routed through the gateway that the server is interacting with. If malicious, this gateway can pretend to be doing so in order to obtain the reward the server has to offer as well as be able to selectively filter traffic to the server. Thus, this verification from the server is not only of the fact that the gateway routes the attack packets, but also that it routes sufficiently many of them and that each of these packets indeed go through the gateway's firewall before being forwarded to its next destination. Moreover, the source address on the attack packets may be fake (spoofed by the adversary) in which case the server needs to know if the gateway is indeed an intermediate node between the true attacker and the server, before any protocol can be run. The authors are currently unaware of any scheme that establishes such *proofs-of-sufficient-routing* and thus, propose this as an open problem introduced by *global*-SDN. The main roadblock here is the anonymity between the server and the gateway prior to this step. The adversary can easily launch a man-in-the-middle attack at this point to fool the server of gateway's routing information.

3.1.4 Handling Collusion. An attacker with access to many gateways could manipulate them to block benign traffic destined to a server, again causing a DoS attack, this time against a server. Furthermore, a natural consequence of a gateway's rationality with respect to the reward is the possibility of a collusion between the gateway and the attacker, where the two join hands in simulating an attack for the sole purpose of extracting the reward from the server. From the server's point of view, the incoming stream of attack packets is indistinguishable from the case when no collusion happens. This makes the server vulnerable to paying money for a fake (simulated) attack. We suspect that an exact detection may be impossible since the gateway can pretend to be honest during the detection phase and fool the server into believing that no collusion in taking place. Thus, techniques like Honeypots [24, 27] or other incentive-driven heuristics may need to be deployed here to catch any suspicious activity and blacklist gateways that are likely to be compromised.

3.1.5 Same Gateway Multiple Attacks. How should the server react if another attack from the same gateway is detected? Should it run the whole protocol again? If yes, how many times should this be allowed? One solution is to adopt a strategy that if a gateway seems to be a source of too many attacks, then the server blocks all communication from that gateway. This seems to be a practical solution, however, if the gateway is honest, the attacker can deliberately make the gateway fall victim to server's blacklisting by continuously routing attack packets of varying nature through this gateway. Even if we assume that an optimal routing policy, which is difficult to tamper with, is in place, the adversary can selectively corrupt middleboxes to maximize its chance of curtaining all communication between a given gateway and the server [11]. The wide variety of new attacks, thus introduced, provide a big room for improvement here to handle the different ways the adversary can (mis)behave.

3.2 Computational Challenges

Apart from the security challenges discussed above, one must take into account the fact that security gateways on the Internet are very limited in their computational power and energy budgets. Moreover, *global*-SDN, based on what was discussed above and what will follow, requires solution to complex computational problems, performing which with sufficient accuracy in real time can be challenging. At the heart of remote protection, lies the ***Remote Policy Learning Problem***, where a remote entity (the gateway) tries to learn a policy to thwart attacks for another remote entity (the server). In the discussion that follows, we highlight some important aspects of this problem and the challenges they pose.

3.2.1 Detection at server's end. The first task in *outsourcing* attack prevention is to discern an impending or an ongoing attack. This would require running an attack detection routine at all times at the server's end. Such a functionality is commonly provided by Network Intrusion Detection Systems (NIDS) that analyze incoming network traffic with respect to anomalies and bad traffic patterns [20, 22, 26]. However, we argue that *global*-SDN does not require the full functionality of an NIDS. All that is sufficient is for the system to be able to trigger a packet logging mechanism in which all incoming packets from a (list of) suspected source(s) are tagged as *good* or *bad* w.r.t. some pre-defined *normal* behavior and stored locally. In this regard, the server needs to have a precise way to classify individual packets as malicious or benign.

There are many such mechanisms for detection, either proposed or already deployed in practice. For example, one can use information-flow tracking to detect code-injection attacks [9, 12], strict control-flow integrity to detect ROP-style attacks [10, 17, 30], and runtime analysis and symbolic execution to detect malware variants [5, 28]. What is common to all of these mechanisms is that they trade off performance for accuracy, making them too slow to run in real time and too resource intensive to process traffic at line-rate speeds. The idea is to use perfect detectors to identify *some* of the attack traffic at the server, and then to use a less-than-perfect but efficient detector to stop *all* of the attack traffic. Needless to say, the design of both *(near) real-time perfect detectors* for the server and *sufficiently-accurate interactively-constructed approximate detectors* for the gateway is an active open problem.

3.2.2 Remote learning of traffic policy. Any model-generation algorithm that filters attack packets from benign packets will require both positive and negative examples to create a sufficiently accurate boundary without over-fitting or under-fitting. Even with the attack packets known, it is entirely likely that the training set is imbalanced, with a larger variety of benign packets than attack packets, making the choice of a benign training set crucial. Two important concerns arise here: (1) How to represent the packets in a way that does not make the server send too many packets to the gateway before a correct model is built? Moreover, it is important to ensure that the set of training and test examples along with the order in which these will be sent to the gateway is supportive of the learning. These considerations are necessary because the server has no good estimate of how complex the learning problem here is and how many rounds it will need to achieve its tolerance setting; (2) How to handle (the difference in) format and semantics of the policy-change request as well dealing with rule generation and filtering of encrypted packets. Different gateways have completely distinct filtering semantics, distinct languages for specifying the filtering policies, and distinct levels of expressiveness for capturing a certain filtering functions. Exposing this information to anyone on the Internet, beyond the challenges of actually representing this information in a common format that everyone agrees upon, is also a security risk for the security gateway, as attackers can study the filtering languages, its expressiveness, and its semantics to find weaknesses that would allow evasion attacks. Moreover, limiting the learning to a particular encryption scheme with make the solution restrictive in practice and leave it prone to attacks.

3.2.3 *Policy Inference from Encrypted Information.*

A big challenge in deriving any traffic policy from network packets, specially on the Internet today, is the fact that most communication is encrypted [13]. One might think that a lot of information can be derived from meta-level features like arrival time and source and destination addresses, it is needless to say that a significant amount of information can be learned if the contents of the packets can be taken into account. Consider the case of phishing emails for example, which may appear to be coming from a legitimate sources but almost always are drafted poorly and can be easily detected by even a mildly aware user. However, unless the recipient of the packets have sufficient information to obtain the contents of the packet, these contents cannot be used to train the traffic policy. If one thinks deeply about it, the whole point of secure encryption is to hide information from an eavesdropper and a sufficiently strong encryption scheme will guarantee that the encrypted messages reveal close to no information at all about the underlying contents. Thus, learning strong filters from encrypted network packets can be challenging at the gateway's end (as compared to the server's end, modulo the side effects of doing so).

3.2.4 *Total Number of Filters.*

For every server that contacts a given gateway for an attack, should the gateway deploy a different rule for each of these requests? If yes, then this is an indirect DoS attack on the gateway itself and its firewall, which will likely not be able to scale up to so many requests. However, if not, then how does the gateway decide the number of rules to deploy in order to maximize the filtering accuracy as well as the reward collected? This is one of the reasons why it is important to keep open the choice of learning algorithm for the gateway, since the gateway can choose to build rules that, instead of serving the request by only one server, cater to a set of servers together. This is equivalent to developing rules that perform multi-class classification over the space of network traffic, instead of the binary case of separating attack from non-attack.

3.2.5 *Real-Time Learning.*

It is important for any protocol that outsources traffic policy generation in real time to be *pro-actively efficient* in that when the filter is ultimately learnt and deployed, it can be used to thwart an ongoing or impending attack. If the learning is too slow or the security verifications take too much time, then it might be too late by the time the protocol terminates. Specially in the cases of an ongoing DoS attack, when the server is already busy in handling the DoS packets and has limited computational power and bandwidth capability to carry out the protocol instructions for remote protection, it is imperative to ensure that the protocol itself does not cause an unintentional DoS on the server and cause it to completely shutdown. If the situation is too severe, care must be taken to abort the *global*-SDN protocol and instead take measures local to the server to stop the attack as soon as possible. Such a decision making is critical and at the same time, difficult to include as part of a fully autonomous system. Nevertheless, it is an important security as well as computational constraint that must be taken care of.

3.2.6 *What should the filters deployed NOT do?*

Among the many scenarios conceivable in which the deployment of a remote filter can go wrong, a fundamental and critical security requirement that one needs to take care of is that the adversary must not be able to weaken or eliminate the current filtering policy of the gateway through careful attack packet design, leaving the local network open to external attacks. In other words, the server could ask the gateway to learn a model that negatively affects the filtering already provided by the existing policies in its firewall. For example, a malicious update to the policy could prevent the local network from accessing the Internet, effectively causing a denial-of-service attack. This must be an unlikely scenario in any solution for *global*-SDN. Furthermore, policy updates must apply only to traffic destined for the server requesting the policy change. This is important to ensure that an adversary colluding with a server is unable to block traffic to selected nodes on the Internet.

3.3 Feasibility Challenges

Predicting the behavior of nodes on the Internet is seldom an easy task and in many cases, it is practically impossible. While certain standard assumptions on the nature of faults and common attack patterns can be argued, a lot of adversarial interventions which are motivated by financial incentives and other unexplained reasons can remain unaddressed. An important issue that now arises in *global*-SDN is the ***Deployment Verification Problem***, which tries to find mechanisms to ensure that the gateway deploys the correct traffic policy in its firewall as it promised to the server and does not lie about what it is filtering and what it is not. The gateway can learn a lot of information about the server's incoming traffic if care is not taken during learning and this information can open many possibilities for a gateway to maliciously craft a filter for the outbound traffic to the server (or no filter at all). From the server's point of view, a provable security guarantee is needed. In the discussion that follows, we highlight some important aspects of this problem and the challenges it poses.

3.3.1 *Assumptions on Network Topology and Message Routing.*

One needs to assume that all traffic from the adversary-controlled node(s) to the Internet passes through (at least) one gateway. Most if not all servers have one or more firewalls locally deployed (from IP firewalls to application-level firewalls), so this assumption is straightforward to satisfy. If the server wishes to deploy the detection policy further away from its immediate network, then the server may have to perform the policy-setting protocol with multiple gateways, one for each network that is a source of attack traffic. Furthermore, it is important to assume that there are multiple (at least two: the server's firewall and a gateway closer to the clients) gateways on the path from each client to the server.

Best practices on the client and the server side often lead to gateways at the boundary between the Internet and the client's and server's local networks, thus most paths would have at least two gateways on them. Links between autonomous systems (ASes) are often suitable points for deploying traffic-control gateways and recent analysis [14] showed that the average distance between ASes is 3.68, meaning that traffic travels over three links, i.e., three suitable control points. A server would then have on average 5 gateways from which to choose for policy deployment on any given Internet path.

3.3.2 *Deployment verification.*

When the server offloads its security enforcement to gateways elsewhere on the Internet, a malicious gateway could pretend to enforce the new policy while still allowing attacks to pass through, exposing the server to exploitation. A naive solution is for the gateway to provide some sort of a *proof*, cryptographic or otherwise, that it has deployed the model in its firewall. One way to do this is to use trusted hardware to ensure that the gateway follows the protocol instructions and no tampering with the code can be successful [7, 8, 21]. A solution like this will be able to *digitally sign* all the future packets from the gateway in a way that reflects the change in the contents in the memory at the gateway's end. However, such a solution is expensive in terms of requiring gateways on the Internet to use trusted hardware, which can be an overkill for the purposes of *global*-SDN. Moreover, since the server has no way to determine the actual contents of the filter deployed by the gateway, any guarantee provided by the trusted hardware is only *w.r.t.* the change in the memory contents but does not ensure that the change was made through the filter that the gateway learned. Finally, if the server requires this service for a prolonged period of time, the gateway has no incentive to retain the model in its firewall. The reward it has obtained for learning is only compensating for the computational resources spent so far. In the best of our knowledge, this is a novel problem and is currently unsolved.

3.3.3 *Slow responsiveness under DoS attack.*

It is easy to see that a protocol for *global*-SDN can be less responsive when the server is under a severe (denial of service) DoS attack. If the attack packet frequency is too high, the server may not be able to communicate with the gateway at all in order to carry out the required message exchange. In this situation, the server should resort to conventional methods to stop the attack. However, detection of this situation at the server's end is crucial before it starts executing any recovery procedure. In an ideal case, *global*-SDN should be predictive enough to thwart an attack of such scale *before* it severely limits what the server can do.

4 TWO USE CASES

We visit two past attacks of global Internet scale, and use their reported scale and impact on particular organizations in order to estimate the feasibility of using *global*-SDN as a defense. At a minimum, the cost of a *global*-SDN-powered defense should be lower than the cost incurred in these past attacks.

Mirai botnet's DDoS attack on Dyn Inc. Let us consider the Mirai botnet and its attack against the DNS provider Dyn Inc. on October 21, 2016. Analysis by Antonanakis et al. [2] show that at least 107,000 IoT devices infected by the Mirai malware launched several rounds of distributed denial-of-service (DDoS) attacks against Dyn Inc.'s DNS servers, results in outages of DNS resolution for a number of popular websites, including Amazon, Github, Netflix, PayPal, Reddit, and Twitter. The attack and its effects lasted approximately 17 hours.

In a *global*-SDN deployment, Dyn Inc. would have identified the networks that contained the most infected devices and then interacted with gateways for those networks to get them to block attack packets. In the Mirai DDoS attack on Dyn, this would mean creating policies that identify the TCP SYN packets originally destined for port 53 of Dyn's DNS servers. As the Antonanakis et al.'s analysis reports, the Mirai botnet exhibited a concentrated network distribution, where the top 10 ASes hosting Mirai-infected IoT devices accounted for 44.3% of infections. Thus, creating and deploying policies for 10 gateways via global SDN could have eliminated about 40% of the DDoS volume.

The Mirai botnet used some 107,000 IoT devices to launch a DDoS against Dyn Inc.'s DNS hosting operations, and in the process leaving major websites such as Amazon, Twitter, and Reddit temporarily unreachable via their domain names. Measuring the cost of attacks is more of an art than a science, and we face the same challenge in this attack. One estimate finds "up to $100 million in [lost] revenue"[1], another that Dyn Inc. lost about 4% of its customers after the attack[2], resulting in a $3.96 million revenue loss (based on Dyn Inc.'s 2015 revenue[3]).

If we rely on the $3.96 million number, then we can estimate an upper bound on the payments Dyn Inc. would be willing to make to various Internet gateways to mitigate the attack. To reduce the DDoS volume by 40% by negotiating with 10 *global*-SDN gateways (see **??**), one would be willing to spend up to $1.58 million in total or about $158,000 per gateway. Whether the gateways are interested in performing the DDoS filtering task at that price point is unknown at this point. It is interesting to note that an earlier victim of Mirai's DDoS attacks, Brian Krebs's website, was protected by Akamai's anti-DDoS service at the estimated cost of $150,000 per year[4], and that Akamai stopped providing this protection when the attack intensified.

WannaCry ransomware infection of UK's National Health Service (NHS) systems. WannaCry is a ransomware that spread around the world in mid-2017, infecting machines vulnerable to a SMB1 bug, encrypting their files, and asking for a ransom to be paid. Observations of the spread showed that millions of machines were infected, across 9,500 unique ISPs[5], and 100 countries[6]. One of the hardest hit organizations was UK's NHS system, where 45 hospitals were disrupted. The cost to the NHS was reported to be £180,000[7].

This is a scenario distinct from Mirai since it is not a denial-of-service attack at the network level, and thus the defense policy to be deployed at a security gateway is not related to the volume of certain type of traffic. The most likely defensive option here is to identify the SMB1 exploit packets that result in deploying the ransomware payload on a victim system. Indeed, there are IDS rules that simply identify the exploit traffic[8] and given the simplicity of these rules, we can presume that automatically inferring such rules from examples is feasible. Then *global*-SDN provides a suitable for defense for NHS against WannaCry.

On the question of costs, a practical estimate is to pay 9,500 security gateways (one per ISP) at most £18 each to learn and enforce rules to stop the exploit. This payment is much smaller that the Mirai DDoS one, but it may be proportionally appropriate given that the amount of work per gateway is much lower compared to Mirai's DNS flood. If this amount of money is insufficient for a gateway to participate in the *global*-SDN protocol, the NHS server(s) may need to prioritize with which gateways to work so that the server can offer a higher reward.

5 CONCLUSION

Motivated by the scarcity of trust on the Internet and the increasing sophistication in the nature of attacks seen today, in this paper we introduced the problem of a fully-decentralized remote attack prevention scheme, which we refer to as Global Software-Defined Networking (*global*-SDN). At the heart of this proposal is a protocol for interactive learning of traffic policies to help a victim of an ongoing or an impending attack to remotely configure a security gateway to install a policy to stop attack packets passing through its firewall. Since a general solution to this problem is currently unavailable, we highlighted many challenges that need to be addressed while designing an Internet scale solution and point out new and exciting open problems that arise. Combining insights from distributed computing, game theory, secure computation and advances in machine learning, we aim to trigger an interest in research in this direction.

[1]http://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html
[2]https://securityledger.com/2017/02/mirai-attack-was-costly-for-dyn-data-suggests/
[3]https://www.inc.com/profile/dyn
[4]https://www.theregister.co.uk/2016/09/26/brian_krebs_site_ddos_was_powered_by_hacked_internet_of_things_botnet/
[5]https://blog.kryptoslogic.com/malware/2017/05/29/two-weeks-later.html
[6]https://www.endgame.com/blog/technical-blog/wcrywanacry-\ransomware-technical-analysis
[7]https://www.digitalhealth.net/2017/07/wannacry-\emergency-measures-cost-central-nhs-agencies-180000/
[8]https://logrhythm.com/blog/using-netmon-to-detect-wannacry-initial-exploit-traffic/

REFERENCES

[1] [n. d.]. https://projectshield.withgoogle.com/public/. ([n. d.]).
[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110.
[3] Mohamed Boucadair, Reinaldo Penno, and Dan Wing. 2013. Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF). RFC 6970. (July 2013). https://doi.org/10.17487/RFC6970
[4] Stuart Cheshire and Marc Krochmal. 2013. NAT Port Mapping Protocol (NAT-PMP). RFC 6886. (April 2013). https://doi.org/10.17487/RFC6886
[5] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. 2005. Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 32–46.
[6] Matthew Connor. 2006. Automated Method for Self-Sustaining Computer Security. (May 10 2006). US Patent App. 11/382,530.
[7] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.
[8] Victor Costan, Ilia Lebedev, Srinivas Devadas, et al. 2017. Secure processors part II: Intel SGX security analysis and MIT sanctum architecture.

Foundations and Trends® in Electronic Design Automation 11, 3 (2017), 249–361.

[9] Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. 2009. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 88–106.

[10] Xinyang Ge, Hayawardh Vijayakumar, and Trent Jaeger. 2014. Sprobes: Enforcing kernel code integrity on the trustzone architecture. *arXiv preprint arXiv:1410.7747* (2014).

[11] Geoffrey Goodell, William Aiello, Timothy Griffin, John Ioannidis, Patrick D McDaniel, and Aviel D Rubin. 2003. Working around BGP: An Incremental Approach to Improving Security and Accuracy in Interdomain Routing.. In *NDSS*, Vol. 23. 156.

[12] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. 2006. A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Vol. 1. IEEE, 13–15.

[13] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. 2015. TLS in the wild: An Internet-wide analysis of TLS-based protocols for electronic communication. *arXiv preprint arXiv:1511.00341* (2015).

[14] B. Huffaker, M. Fomenkov, and k. claffy. 2016. *Statistical implications of augmenting a BGP-inferred AS-level topology with traceroute-based inferences - Technical Report*. Technical Report. Center for Applied Internet Data Analysis (CAIDA).

[15] Sam Jones. 2017. Timeline: How the WannaCry cyber attack spread. *Financial Times* (May 2017). https://www.ft.com/content/82b01aca-38b7-11e7-821a-6027b8a20f23.

[16] Ethan Katz-Bassett, Harsha V Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter Van Wesep, Thomas E Anderson, and Arvind Krishnamurthy. 2010. Reverse traceroute.. In *NSDI*, Vol. 10. 219–234.

[17] Mehmet Kayaalp, Timothy Schmitt, Junaid Nomani, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2013. SCRAP: Architecture for signature-based protection from code reuse attacks. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 258–269.

[18] Ryan Kearny. 2017. Of Needles And Haystacks: Spotting Bad Network Traffic In The Age Of Botnets. *Forbes* (June 2017).

[19] J. Knight. 2004. Method and system for remote network security management. (Dec. 16 2004). https://www.google.com/patents/US20040255167 US Patent App. 10/834,443.

[20] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, 1 (2013), 16–24.

[21] David Lie, Chandramohan A Thekkath, and Mark Horowitz. 2003. Implementing an untrusted operating system on trusted hardware. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 178–192.

[22] Teresa F Lunt et al. 1989. Real-time intrusion detection.. In *COMPCON*. 353.

[23] Zhuoqing Morley Mao, Jennifer Rexford, Jia Wang, and Randy H Katz. 2003. Towards an accurate AS-level traceroute tool. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 365–378.

[24] Marcin Nawrocki, Matthias Wahlisch, Thomas C Schmidt, Christian Keil, and Jochen Schonfelder. 2016. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249* (2016).

[25] K. Psounis. 1999. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys* 2, 1 (First 1999), 2–16. https://doi.org/10.1109/COMST.1999.5340509

[26] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad hoc networks* 11, 8 (2013), 2661–2674.

[27] Anjali Sardana, Krishan Kumar, and Ramesh Chandra Joshi. 2007. Detection and honeypot based redirection to counter DDoS attacks in ISP domain. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*. IEEE, 191–196.

[28] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and privacy (SP), 2010 IEEE symposium on*. IEEE, 317–331.

[29] Mark Silverstein. 2018. The New Life of SmartNICs. Published online at https://www.sigarch.org/the-new-life-of-smartnics/ . Last accessed: 22 April 2018. (April 2018).

[30] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2013. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 574–588.

[31] Dan Wing. 2011. Port Control Protocol. *The Internet Protocol Journal* (dec 2011).