X-Pro: Distributed XDP Proxies against DDoS

Syafiq Al Atiiq
syafiq_al.atiiq@eit.lth.se
LTH, Lund University
Lund, Sweden

Christian Gehrmann
christian.gehrmann@eit.lth.se
LTH, Lund University
Lund, Sweden

Abstract

The concept of Internet of Things promised to connect billions of devices around the globe to the internet and capable to communicate each other. However, the device itself is especially vulnerable to be used as a bot to launch a massive Distributed-Denial-of-Service attack. The main problem with the IoT devices are usually resource-constrained, such that implementing a strong security protection is a non-trivial problem. Hence, they are easily turned from a usable embedded device into a bot for launching DDoS attack.

A source-based DDoS detection could help reducing the impact of DDoS attack by blocking traffic from the source address of adversary. However, determining the legitimacy of the traffic close to the source is a hard problem, because the volume might not be big enough to be deemed as an attack. Also, from the economic standpoint, putting a classifier node close to the source of an attack is not incentivise the service provider. In this paper, we present a reverse-firewall proxy on the cloud working together with virtualized network interface of the embedded device to mitigate against DDoS attack. On the IoT side, our solution can be deployed by pure implementation on the network interface function of embedded device, not affecting the rest functionality of the device itself. Also, as the DDoS detection happens at the cloud, it is possible to share a large number of real-time traffic situation to make a better decision on the legitimacy of a traffic. We evaluate the performance of our solution from several point of view. In conclusion, our solution offers a better decision making in terms of an attack traffic, as well as preserving the resource of the victime from a severe DDoS attack.

Keywords reverse-proxy, internet of things, denial of service, security

Gwangju '21, March 22–26, 2021, Gwangju, Korea

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Syafiq Al Atiiq and Christian Gehrmann. 2020. X-Pro: Distributed XDP Proxies against DDoS. In *Proceedings of SEC@SAC '21: The Security Track at the ACM Symposium on Applied Computing (Gwangju '21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

From September 2016, a tremendous distributed denial of service attack launched against several high-profile website on the internet, namely OVH [**?**], Dyn [**?**], and Krebs on Security [**?**]. Surprisingly, the source of the traffic came from a huge amount of the internet of things devices turned into bots. These bots controlled by a master process, which later known as Mirai botnets [**?**]. The master process scanned the whole internet and infects embedded devices that still running with insecure default password.

Aside from the fact that people do not change their default password in their IoT devices, one of the key problem in this situation is that many internet of things devices cannot afford to implement strong security protection in terms of software and hardware features. Thus, they are considered as easy targets for an adversary who wants to utilize them as a weapon to launch DDoS. As the number of these small devices are only increasing, the magnitudes of the attacks will only get increased. Based on the Krebs on Security measurement [**?**], the initial attack of the mirai botnets reached 600 Gbps, one of the largest attack on the internet so far.

Even worse, as most of the internet of things devices are typically resource constrained, the device might loose lots of performance due to undesirable energy consumption. From one point of view, this is a problem which need to be solved. However modest, in this paper, we put more of our focus on how to reduce the impact of the attack on the victim side even when the embedded devices are compromised, i.e. infected by the bot software.

To fight against different types of DDoS attack, a number of countermeasure procedures have been proposed. Based on [**?**], there are two types of defense mechanism against DDoS, namely (i). destination based, and (ii). source based. Our work in this paper fell into the latter category. In the context of source-based defense mechanism, some of the existing work includes, for instance: D-WARD [**?**], MULTOPS [**?**], and MANAnet Reverse Firewall [**?**].

D-WARD [**?**] monitors the inbound and outbound traffic in the perspective of the flow, which then compared to the

normal flow. If there is any suspicion to a specific flow, then it gets blocked. D-WARD should be deployed at the border of the source network to be able to stop an attack. Based on [**?**], even though the filtering mechanism can happen at the source, D-WARD still consumes more memory space and CPU cycles compared to recent network-based defense mechanism. Also, since the deployment is close to the source, there is no incentives for the network provider to deploy D-WARD as it supposed to protect others network, not the one who actually deploy D-WARD. As there is little to no economic incentives, it might be a bit hard to justify the cost of deployment.

MULTOPS (multi-level tree for online packet statistics) and TOPS (tabulated online packet statistics) [**?**] employs a tree data-structure and capturing the rate statistics from the various subnets. The assumption is that traffic between two hosts is roughly proportional. Hence, when the traffic is not proportional, then one of the two hosts is either a source or the destination of an attack. However, not all legitimate traffic should behave proportionally when it comes to incoming and outgoing comparison. For example, a large multimedia stream from multiple sources does not have a proportional traffic. Most of the times, the traffic originating from clients are much smaller compared to the traffic from the server. To this end, MULTOPS and TOPS can have an immense number of false negative flag.

To fill the gap, in this paper we introduce a reverse firewall proxy to counteract DDoS attack which has specific source from the embedded devices turned into bots. In particular, we leverage the use of scalable proxies in the network [**?**], with a virtualized wireless interface [**?**] together with load balancing optimization. But in our case, this optimization is not designated to increase the performance of the network, but rather for DDoS reverse firewall decision making. Our solution can be deployed on the embedded device on the network interface stack, not affecting the rest of the functionality. Different from prior-art source based DDoS mitigation techniques, our solution does not happen close to the source, i.e. embedded device, but rather at the cloud. This allows sharing of a real-time traffic situation for a very large number of devices at the proxy cloud side. This in turn gives much better possibilities to detect network based DDoS attempts as traffic information about DDoS attack destination targets is available to all proxies.

We have made a proof-of-concept implementation with the following details. On the cloud side, we leverage eXpress data path [**?**] running on a baremetal hardware. It is possible to run our software on the higher stack in the operating system. However, we try to get a better performance by having only the packet processor down in the kernel-space. On the embedded device side, we have made an implementation on top of pycom platform [**?**]. Fipy [**?**], connected to an NB-IoT network, acting as the virtualized network interface. This comes without having to modify the firmware or software architecture in the main MCU. Also, as the protocol itself is general, it can be implemented on any platform, without the necessity to be run on the same hardware. As the proxy allows real-time information sharing in regards to the traffic condition, our protocol offers better ability to detect a possible DDoS attempt from a broader perspective. Our results show that the impact on the victim can be reduced significantly, even during a massive DDoS attack.

The rest of the paper is organized as follows. We discuss related works in Section II and background concepts in Section III. We provide the application scenario in Section IV. Section V presents scalable reverse firewall proxy, while in Section VI we provide a performance evaluation. Section VII draws our conclusions and anticipate future works.

2 Related Work

A major security problem in the current IoT networks is Distributed Denial-of-Service (DDoS) attacks where the IoT units are used as bots. In particular, we consider the problem of source based DDoS in the context of Internet of Things. It is considered as a non-trivial problem, since the source of the attacks can be distributed in different domains making it difficult for each of the source to detect and filter attack flows accurately. Also, it is difficult to differentiate between legitimate and attack traffic near the sources, since the volume of the traffic may not be big enough as the traffic typically aggregates at points closer to the destinations.

To counteract DDoS harnessing IoT as bots, there have been several approach proposed. In this section we take a closer look on what is already available solution and how they work. The first approach is machine learning based solution, where the idea comprises a machine learning in a way or another. Based on [**?**], the ML based solutions are classified into the following techniques : (i). Using IoT network behavior, and feature extraction [**?**]. The author claims that using IoT-specific network behaviors, i.e. finite number of endpoints and assuming that there is a well-specified interval between pakcets, it is possible to do feature selection with high veracity to detect DDoS in the network. The way the feature selection is performed is by implementing a low-cost machine learning algorithm in the middleboxes, i.e. home-router, local-gateway, switch, etc. However, we argue that if the underlying assumption does not met, we might end-up getting false positive (or negative) in the feature selection.

(ii). Harnessing SDN architecture to perform DDoS detection [**?**]. The authors present how SDN, combined with statistical approach can detect DDoS attempt in the network. The authors emphasizes that the work is not only to classify the bogus messages, but also more importantly, to misclassify the legitimate traffic. While the result shows that it is possible to employ statistical classification along with SDN to detect DDoS, the authors explicitly show that a meticulous consideration needs to be performed to select classifiers

with lowest possible overhead. Otherwise, the computing resource to perform the classification itself will occupy the middlebox and worsen the traffic condition.

Another new approach is by detecting malicious traffic based on the heartbeat association [**?**]. The heartbeat association then used to create heartbeat network, the heartbeat associated graph and the attribute propagation algorithm based on the graph. While the result shows that the method can effectively detect the DDoS malignant host, the computation needs to be performed at the border of the router. Even though the computing resource might not be a problem, but we argue that a much larger resource would be needed when the DDoS attack is getting bigger.

Our solution fills the gap by effectively send all the traffic from the IoT device to our proxy while at the same time maintaining all the proxy well-informed on the traffic condition from a wider perspective. Also, as the informations are shared between proxies, proxy in different locations can catch-up the whole situation faster once DDoS happen in the network. This lead to a better DDoS mitigation in case the master of the botnets decide to launch the attack simultaniously using different bots from different locations.

3 Background

3.1 Express Data Path

Express Data Path (XDP) [**?**] is a novel programmable packet processing, living inside the kernel-space. Currently, XDP is part of the mainland linux kernel and has the ability to work together with other parts of networking stack in the kernel.

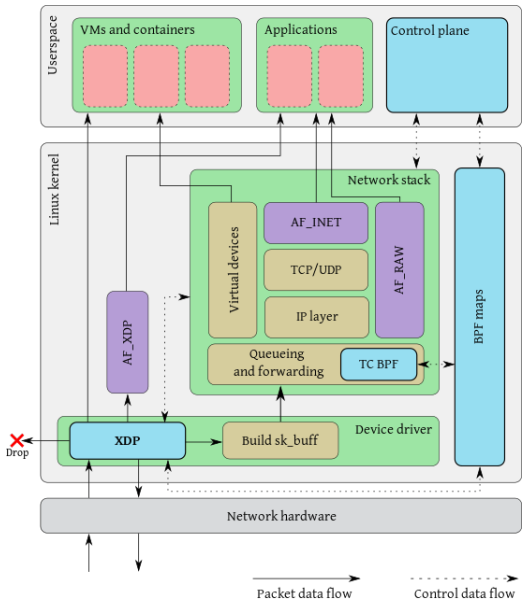

Figure 1. XDP Building Blocks [**?**]

As shown in fig 1, when the packet arrives, there is an XDP hook inside the device driver. It has the ability to run an eBPF program, such that the XDP hook could determine whether to:

1. Drop the packet
2. Send the packet back out to the same interface
3. Redirect the packet, either to the same or different interface
4. Forward the packet to the userspace program, or
5. Allow the packet to be processed through a normal networking stack

Our aim in this paper is to block the packet as early as possible, i.e. in the XDP hook, and communicate the control plane data, i.e. which packet should be dropped, through the BPF maps. A more detailed description about how we utilize XDP is explained in the next chapter.

The logic in XDP is written in high level language, i.e. C, and compiled into a bytecode. Kernel has the job to statically analyze the produced bytecode, in terms of the safeness before the execution. If the code is safe, then the bytecode will be translated into a native instructions. We leverage XDP simply because of its performance. It has been proven that a single-core CPU running an XDP packet processor software could process as high as 24 million packets per second.

4 Reverse-Firewall-Proxy

4.1 Proxy system architecture

The proxy system consists of a set of proxies interconnected through an internal IP network. Each proxy is also assumed to have direct network connectivity. The proxy nodes shares filtering and also load information using a shared DB in the system. The solution is agnostic to a particular DB sharing method, but in our case, we use an in-memory database called Redis [**?**]. A wholistic perspective of our proxy architecture can be seen in the fig 2.

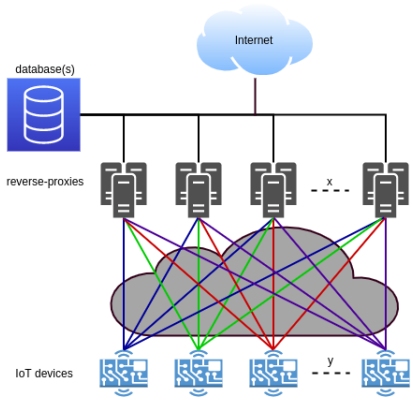

Figure 2. Mesh Network between IoT Devices and Reverse-Proxies

Each IoT unit must have the connectivity towards all available proxy. It means, mesh network will be created between

IoT units and the respective proxy. Also, as already mentioned previously, all proxies must have the same visibility in regards to which message needs to be blocked. Therefore, there has to be a mechanism to share information about this between proxies one way or another.

As we would like to block the bogus messages as early as possible in the XDP hook, those information needs to be shared between kernel-space and user-space. This is where the BPF maps comes into place. Each proxy has a running redis instance, which syncronize each other through redis replication mechanism [**?**], performing the synchronization protocol explained in subchapter 5.2. The syncronized data then communicated to the eBPF program running in the XDP hook through the BPF maps. This way, we are able to pass the information between proxies without the need to send the invalid packet to the userspace. This way, we can reduce the CPU utilization in the userland and allocate the CPU into more important task, i.e. packet filtering in the kernel. Our argument is that the less tasks performed in the userspace, the more CPU can be utilized by the XDP to block the invalid messages, hence we get more packet filtering capacity in the kernel. Figure 3 represent the connection between our packet filter mechanism with the syncronization function between databases.

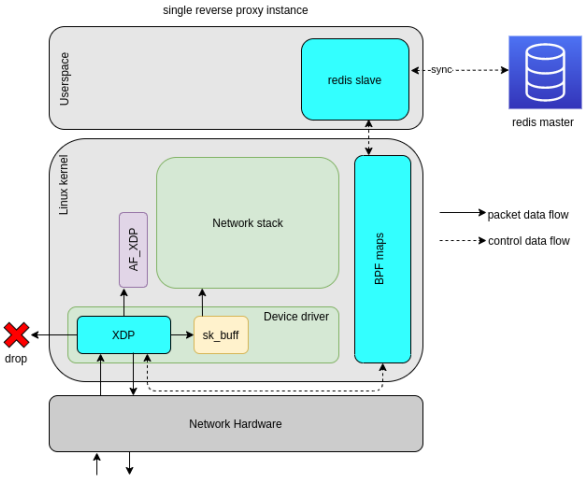

Figure 3. Syncronization between proxy and master DB

4.2 IoT unit architecture and general procedure

From the standpoint of IoT device, the proxy management is purely handled by the network interface or modems. This approach allows the modem to provide a standard network interface to the IoT OS. Figure 4 depict the relationship between IoT main MCU with the network interface.

The network modem is responsible for the proxy management for outbound traffic, i.e.:

- The network modem keeps track of available IoT proxies in the system and change âĂIJcurrent destination proxyâĂİ if the one currently use is overloaded or unavailable. Information of availability and load is obtained through a separate proxy management protocol, explained in the next subchapter.
- The modem tunnels all outbound traffic to the current proxy using either an IP tunnel, http or CoAP tunnel where the original source IP packet is encapsulated (not preventing IP or http level end-to-end security). Inbound traffic is treated completely transparently not affecting the modem or the IoT system at all.

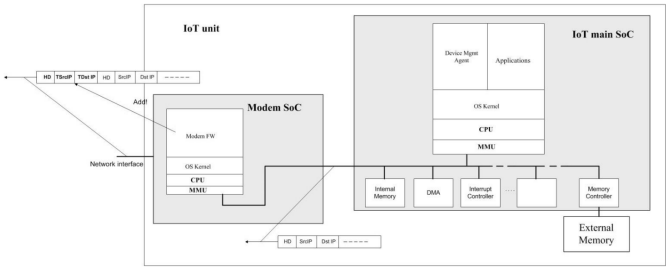

Figure 4. Dedicated Tunneled Network Interface (**a better and clearer picture will be created later**)

5 Algorithm

5.1 Notations

- Set of IoT devices in the system: U
- An IoT device: $u \in U$
- Set of proxies devices in the system: P
- Proxy: $p \in P$
- A device unique index given to an IoT device: i
- An IoT device with index i: u_i
- A proxy unique network address associated with a proxy: j
- A proxy with address j: p_j
- A symmetric shared secret between IoT unit u_i and proxy p_j: K_{ij}
- An integrity protection session key shared between an IoT unit and a proxy: IK
- An encryption protection session key shared between an IoT unit and a proxy: EK
- Work load of a proxy unit: W
- Work load threshold of a proxy unit: T_W
- Time stamp indicating the "oldest time" packet time for a particular (i, D_{addr}) pair: ts_1
- Time stamp indicating the "most recent" packet time for a particular (i, D_{addr}) pair: ts_2
- Packet counter or a particular (i, D_{addr}) pair: c
- Delta packet counter (internally within a proxy) counter for a particular (i, D_{addr}) pair: dc
- First filtering reset threshold used by a proxy: T_{T1}
- Filtering minimum measure time threshold used by a proxy: T_{T2}

- Second filtering minimum measure time threshold used by a proxy: T_{T3}
- Fourth filtering minimum measure time threshold used by a proxy: T_{T4}
- First packet maximum allowed frequency threshold used by a proxy: T_{F1}
- Second packet maximum allowed frequency threshold used by a proxy: T_{F2}
- Frequency division factor: r^2
- Destination IP address of a packet: D_{addr}

5.2 Proxy Synchronization Protocol

For every time period t, each proxy p_j performs the synchronization procedure with the master database, described in the algorithm 1. The initial process started with p_j updates its current workload to the M_{DB}, which then replied by M_{DB} with the information containing the workload for all other proxies except p_j. With those information in hand, p_j looks up each pair (i, D_{addr}) sequentially and performs further computation.

For each pair, p_j looks up at the local database L_{DB}. If L_{DB} contains information about (i, D_{addr}), then further processing is performed according to the algorithm 1. Otherwise, information about ts_1', ts_2', c' will be fetched from M_{DB} and copied to L_{DB}. At the end of the syncronization procedure, p_j updates the remaining information for all (i, D_{addr}) to the M_{DB}.

5.3 Packet Filtering Procedures

When the packet arrives at the proxy, the procedures described in the algorithm 2 takes place. Initially, p_j looks up and read the time stamps ts_1' and ts_2' for the record (i, D_{addr}) from the local database L_{DB}. If the record is found, p_j will compare the difference between the current time and ts_2' with the first filtering reset threshold used by the proxy. If the difference value is bigger than T_{T1}, then ts_1' is set to be the current time, c' is reset to be 0.

On the other hand, if L_{DB} does not contain (i, D_{addr}) p_j is looking for, both ts_1' and ts_2' will be set to the current time, as well as resetting c' to 0. The next step is to increase both the counter c' and the counter difference dc values with 1. If the difference between ts_1' and ts_2' is larger than the second filtering minimum measure time threshold T_{T2}, then p_j compares the value of c' divided by the difference of ts_1' and ts_2' with the first packet maximum allowed frequency threshold, T_{F1}. If its greater than T_{F1}, the respective packet is dropped by the XDP.

Further packet processing performs simillar mechanism to the previous one, but involving $ts_1*, ts_2*, c*, T_{T3}$ and T_{F2} as described in the algorithm 2. If the respective packet passed all the check mentioned above, the tunnel header will be removed and the packet is forwarded to the final destination.

Algorithm 1 Proxy Synchronization Protocol

p_j send the W value to M_{DB}
p_j reads the current workload for all other proxies in the system, $\{W_k\}, p_k \in P, k \neq i$
p_j looks all the pair (i, D_{addr}), for $u_i \in U$
for each (i, D_{addr}) in M_{DB} **do**
  if $L_{DB} \ni i, D_{addr}$ **then**
    if ($mark = 1$ and $ts_1' - ts_1 > T_{T1}$) **then**
      $< mark' = 0, dc' = 0 >$
      $< ts_1 = ts_1', ts_2 = ts_2', c = c' >$
    else
      $< mark' = 0 >$
      if $ts_2' < ts_2$ **then**
        $< ts_2' = ts_2 >$
      else if $ts_2' - ts_1 > T_{T4}$ **then**
        $< ts_1' = ts_2' - (ts_2' - ts_1)/r >$
        $< c = \lfloor c/r \rfloor, ts_1 = ts_1', ts_2 = ts_2' >$
      else
        $< ts_2 = ts_2' >$
      end if
      if $ts_1' > ts_1$ **then**
        $< ts_1' = ts_1 >$
      else if $ts_2' - ts_1' > T_{T4}$ **then**
        $< ts_1 = ts_1' >$
      end if
      $< c' = c + dc', c = c', dc' = 0 >$
    end if
  else
    $< ts_1' = ts_1, ts_2' = ts_2 >$
    $< c' = c, dc' = 0, mark' = 0 >$
  end if
end for
for each (i, D_{addr}) in L_{DB} **do**
  if $M_{DB} \not\ni i, D_{addr}$ **then**
    $< ts_1 = ts_1', ts_2 = ts_2', c = c' >$
  end if
end for

6 Implementation

7 Experimental Evaluation

7.1 Results

8 Conclusion and Future Work

Algorithm 2 Packet Filtering Procedures

<Lookup ts_1', ts_2', c for record (i, D_{addr}) in L_{DB} >
if record found **then**
    if $t - ts_2' > T_{T1}$ **then**
        $ts_1' = t, c' = 0, dc = 0, mark = 1$
    else
        <do nothing>
    end if
else
    $ts_1' = ts_2' = t, c' = 0, dc = 0, mark = 0$
end if
$c' = c' + 1, dc = dc + 1, ts_2' = t$
if $ts_2' - ts_1' > T_{T2}$ **then**
    if $c'/(ts_2' - ts_1') > T_{F1}$ **then**
        <Drop packet>
        <Send an overload warning>
    end if
end if
$ts_1* = \min_{u_i \in U_{D_{addr}}} {ts_1}_i'$
$ts_2* = \max_{u_i \in U_{D_{addr}}} {ts_2}_i'$
$c* = \Sigma_{u_i \in U_{D_{addr}}} (c_i + dc_i)$
if $ts_2* - ts_1* > T_{T3}$ **then**
    if $c*/(ts_2* - ts_1*) > T_{F2}$ **then**
        <Drop packet>
    end if
end if
<remove tunnel header and forward packet>