Visualization of Actionable Knowledge to Mitigate DRDoS Attacks

Michael Aupetit[*] Yury Zhauniarovich[†] Giorgos Vasiliadis[‡] Marc Dacier[§] Yazan Boshmaf[¶]

Qatar Computing Research Institute, HBKU

Abstract

Distributed Reflective Denial of Service attacks (DRDoS) represent an ever growing security threat. These attacks are characterized by spoofed UDP traffic that is sent to genuine machines, called amplifiers, whose response to the spoofed IP, i.e. the victim machine, is amplified and could be 500 times larger in size than the originating request. In this paper, we provide a method and a tool for Internet Service Providers (ISPs) to assess and visualize the amount of traffic that enters and leaves their network in case it contains innocent amplifiers. We show that amplified traffic usually goes undetected and can consume a significant bandwidth, even when a small number of amplifiers is present. The tool also enables ISPs to simulate various rule-based mitigation strategies and estimate their impact, based on real-world data obtained from amplification honeypots.

Index Terms: C.2.0 [Computer Systems Organization]: Computer Communication Networks—Security and Protection; K.6.m [Computing Milieux]: Miscellaneous—Security

1 Introduction

Distributed Denial of Service (DDoS) attacks are not a new phenomenon. In November 2, 1999, the CERT Coordination Center invited 30 security experts to address what was considered at that time as a new category of attack tools, which used distributed systems to run concurrent denial of service attacks. One of the outcomes of that event was a report containing precise information about protecting systems from the attacks launched by these tools, detecting the use of the tools, and responding to the attacks [17]. That same report also provided precise recommendations for various specific groups within the Internet community, including managers, system administrators, ISPs, and incident response teams, informing them which actions they should take in order to protect themselves.

More than 15 years later, not only we still observe the very same DDoS attacks, albeit implemented using new tools, but also a resurgence and a dramatic increase of the number and strength of these attacks. One explanation of this new wave of DDoS attacks lies with the recent rise of the so-called Distributed Reflective Denial-of-Service (DRDoS) attacks. In these attacks, an adversary sends a request with a spoofed IP address of the victim to a genuine server that runs a vulnerable protocol. The server in turn replies to the victim with a much larger response. Using these valid protocols, attackers can flood the victim with up to 500 times the amount of traffic they initially sent to the genuine servers, now referred to as amplifiers.

Despite the daily occurrence of DDoS attacks, most of the Internet users are unaware of their existences. Indeed, it is quite rare nowadays to learn that a major web server was unreachable because of a DDoS attack launched against it. The reason lies not only in infrastructure improvements, such as large replication of servers in geographically diverse environments, but also in the availability of several commercial offerings capable of reacting to the attacks, when they are launched against high-stack targets. While such commercial solutions are effective in practice, they are expensive and available only to those websites that can afford them. Moreover, as shown by several recent studies [18, 31], today's DRDoS attacks target a large population of the Internet users who do not benefit from existing, commercial defenses. Consequently, the Internet traffic is routinely polluted with large quantities of junk traffic contributing to thousands of daily DRDoS attacks.

In this paper, we present a visualization tool that helps decision makers within an ISP to understand how much resources they waste due to DRDoS attacks. The tool also simulates the efficiency and the implications of various mitigation strategies that are available to ISPs. To achieve that, we combine the knowledge of ongoing DRDoS attacks obtained from amplification honeypots, such as AmpPot [31], with the knowledge of the amount of possible amplifiers existing within a given ISP, as provided, for instance, by the Censys project [22]. We explore new visualization paradigms to extrapolate the junk traffic traversing an ISP's external gateway. We propose these visualizations in a way that helps the ISP to find a mitigation strategy, such as access-control rules, that are practical and would result in the highest benefit in terms of preventing or limiting junk traffic that is attributed to DRDoS attacks.

The structure of this paper is as follows. In Section 2, we cover related work and position this work with respect to the novelty of our contribution. In Section 3, we present the overall architecture of the system we have built to collect, enrich, analyze, and make the data available for visualization purposes. Section 4 describes the data we use in the paper. In Section 5, we describe the contributions we have made in terms of visualization, how they have been designed, and how they address the problem of estimating the gain of different DRDoS mitigation strategies in terms of saving network bandwidth. In Section 6, we present two use case scenarios showing how the interface can be used with real world data obtained from our ISP partner. Finally, Section 7 concludes the paper.

2 Related Work

Honeypot and DDoS Analysis. In late 1999, CERT published a report to warn the Internet community about the threat of DDoS attacks, and offered concrete preventive actions to mitigate the threat [17]. A few months later, the Internet was hit by the first, large DDoS attack [26], followed by many others years after. Since then, researchers have analyzed some of the tools used to launch DDoS attacks [20, 19, 21], measured their impact on the Internet [37, 28, 36, 16, 34], and provided a number of defensive approaches [32]. Eventually, these research efforts led to a number of effective and reliable anti-DDoS products that are provided either as independent appliances or as cloud-based services.

Recently, the interest in defending against DDoS attacks has increased due to the proliferation of DRDoS attacks. In order to provide an open environment for researchers to study this new type of DDoS attacks, Christian Rossow started the AmpPot project in 2014 [31, 40, 41], providing a honeypot that acts as a usable amplifier. Rossow et al. used the data collected by AmpPot deployments

[*]e-mail: maupetit@qf.org.qa
[†]e-mail:yzhauniarovich@qf.org.qa
[‡]e-mail:gvasileiadis@qf.org.qa
[§]e-mail:mdacier@qf.org.qa
[¶]e-mail:yboshmaf@qf.org.qa

across the world to measure, analyse, and show new patterns about the victims of DRDoS attacks, which have been confirmed by other independent studies [18, 23, 15, 30].

Some of the interesting findings about DRDoS attacks are that they are perpetrated against gamers, as a form of cheating or for financial gain, and that a large fraction of victims are end-users who cannot afford existing, commercial anti-DDoS services.

Security Analytics Platforms. Besides enterprise security analytics platforms [12, 6, 11, 8, 10], many open-source solutions have been developed supporting a wide spectrum of security-based analysis [9, 2]. For example, Apache Metron [2], the successor of Cisco OpenSOC [9], is a platform that offers a full-stack software infrastructure to analyze and detect network intrusions, zero-day attacks, and advanced persistent threats.

One mutual approach followed by the exiting security analytics platforms is to provide a robust framework based on prominent technologies, including machine learning and graph analytics, to detect today's security threats. This framework typically involves a set of tools to capture network traffic and collect network telemetry, to enrich the obtained data and index in external datastores. In this work, we leverage the same concepts to build our QaCIP platform, which we describe in detail in Section 3.

Visualization. A wide range of visualization techniques have been applied to support visual analysis of network security data [33, 39, 35]. All these approaches introduce and provide novel visualization techniques specifically for security-related data. According to the recent survey by Shiravi et al. [43], security visualization can be divided into five use-case classes: host/server monitoring with Glyphs [24], internal/external monitoring with parallel coordinates [46], port activity visualization with scatter plots [35], attack patterns with colour maps [27], and routing behavior summaries with histograms [44]. However, we are not aware of any visualization techniques that are dedicated for supporting the design of network traffic filtering rules for cyber security applications.

To support visual large data exploration, Murray et al. proposed a query language, called Kaleidoquery, to make database queries using graphical user interfaces [38]. This language, along with a supporting tool, maps SQL statements to graphical elements, providing a visual data navigation technique to select or filter out a subset of the data using graphical representation. Although Kaleidoquery assists the user in understanding the structure of a filter, it does not support visualization of the effects of the produced queries on the amount of queried data. Decision trees can be also used to represent data filtering [45, 25], where a tree represents a filter that is applied to the data passing from the root to the leaves. While decision trees are useful to visualize the data filtered on each level, they are evaluated across the whole tree, which hides the effect of the filter on a particular level in the tree.

To address the shortcomings of Kaleidoquery, Huo et al. proposed KMVQL, a query language in which an interactive Karnaugh table is used to express boolean queries and to visualize their results [29]. In KMVQL, the table occupies all the space to show all possible binary attribute combinations to choose from. In network security data, however, it is likely to have a large number of multi-level attribute, which makes KMVQL inefficient as the size of a Karnaugh table is exponential in the number of binary attributes. In this work, we propose a new metaphor to support visual traffic filter definition that is tailored to large-scale cyber security applications.

3 Architecture

In order to collect, enrich, analyze and make network security data available for visualization purposes, we have developed a cyber security platform, called QaCIP, which facilitates DRDoS attack analysis. Its architecture is illustrated in Figure 1 and described in details below.

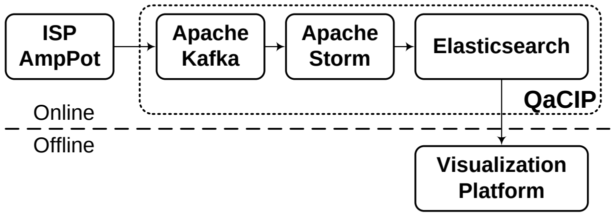


Figure 1: QaCIP analysis platform for DRDoS activities.

3.1 Amplification Honeypots

The QaCIP platform ingests data collected and streamed by amplification honeypots that are deployed by ISPs. This is depicted by the "ISP AmpPot" component in Figure 1. From an ISP standpoint, these honeypots are a valuable source of information for detecting network scanners and amplification attacks [31]. In particular, the honeypots pretend to run services known to be vulnerable to amplification attacks, such as DNS or NTP, so that when they are scanned and exploited by real attackers, the ISP can collect useful information to better understand the intrinsics of amplification attacks and the motivations of the attackers.

Unlike traditional honeypots, amplification honeypots can only be used to analyze data related to the victims, not the attackers. This is the case because these honeypots are not the target of the attack, and the traffic they receive is forged and has the spoofed IP of the victim. This peculiarity allows us to use the collected data to protect ISP networks from the excessive bandwidth incurred by the participation of their innocent client vulnerable servers in amplification attacks.

The main idea behind ISP-driven DRDoS protection is that only attackers, who scan ISP networks to find vulnerable servers, will contact ISP-deployed amplification honeypots. These attackers also know about other, real vulnerable servers in the network, but ISPs cannot simply block traffic to vulnerable servers, as this may harm users making legitimate requests. Instead, an ISP can analyse the data collected by amplification honeypots, and then construct and enforce network access control rules to block the traffic from vulnerable servers to the victims, effectively suppressing the amplification while not blocking the service.

In our setup, we used AmpPot [31] as an amplification honeypot. Unfortunately, the default AmpPot implementation is not designed for high-volume traffic with real-time log streaming. In particular, the original implementation has several traffic limiting strategies that are used to reduce the amount of stored data. Moreover, the implementation stores the collected data in a database file. This makes the original AmpPot suitable for offline data analysis, but it falls short when integrated in a stream processing pipeline for real-time network data analytics that provides the big picture of network operation. This renders the original AmpPot implementation unsuitable for mitigating ongoing DRDoS attacks.

To overcome these limitations, we modified AmpPot as follows. First, we disabled all request limiting mechanisms so that an ISP is able to inspect data related to all requests coming to the honeypot. Second, we implemented a mechanism to send logs of incoming requests in near real-time to QaCIP analysis platform, as described in Section 3.2. These modifications allows us to see the full picture of ongoing attacks and take the appropriate actions to mitigate them.

3.2 QaCIP Analysis Platform

To analyze the data collected and streamed by AmpPot instances, we have designed a scalable, centralized architecture called QaCIP. As shown in Figure 1, the architecture of QaCIP consists of three key components. First, a reliable data aggregator that is able to collect and stream data from several AmpPot instances. Second, a real-time processing engine responsible for parsing the raw data, enriching them with contextual metadata and parceling the result into messages of a unified format. Third, an external datastore that stores and indexes the structured messages, which can be easily exported for visualization. The key advantage of all three components is that they can easily scale and cope with the increased amount of data received, simply by adding more hardware resources. Below we provide more details about these components.

All incoming raw logs are collected using Apache Kafka [1], which acts as the message broker in QaCIP. In particular, the raw log records from the deployed AmpPot instances are published directly in Apache Kafka. To balance the load, the incoming raw log records are stored in different partitions, which leads to a higher throughput. When publishing a log record, Kafka deterministically maps the record to a partition based on the hash of the key, which in our case is derived from the source IP address. Each raw log record contains several attributes extracted from the network packet, such as the IP and UDP header fields, in addition to a timestamp that is added explicitly by the AmpPot instance at the time of capturing. The timestamp allows us to provide an accurate analysis even in cases of communication latencies or delayed processing due to increased utilization.

The collected raw logs are retrieved and processed at real-time using Apache Storm [3], which uses *bolts* to easily scale-in/out. The bolts perform discrete processing actions, which in our case include: (i) the parsing of the raw data into structured messages of attribute-value pairs, (ii) the enrichment with contextual information, and (iii) the export to an external datastore. QaCIP's data model consists of JSON messages, which define the physical interpretation of data. So, the parsing bolt is responsible for creating a new JSON message that contains the corresponding data of each incoming log record. We mainly use basic types to represent a single value (e.g., booleans, integers, floating-point, epoch times, strings, and IP addresses), and array types for bundled values (e.g., sets and tables). Each newly created JSON message is then passed to the next bolt, which enriches the message with contextual information. For instance, using IP address it is possible to add geo-location information and the ASN, which IP belongs to. This allows us to provide a human-understandable network view of the raw data, and be able to efficiently answer popular operational queries, such as which cities or countries are under heavy attack. The enriched JSON messages are fed to the last bolt of our topology that is responsible for exporting them to the external datastore. The three bolts are connected sequentially, and form a topology pipeline that is executed independently for each incoming log record, in parallel. Parallelism can be easily increased by creating the required number of threads for each bolt.

A major design decision is how to group the incoming log records and pass it from one bolt to the next one. Since our processing is stateless, the naive way is to shuffle the incoming log records in a round-robin fashion. This will allow a harmonized and equal load balance across different threads of the corresponding bolts. However, it will result in poor performance when enriching incoming records with the metadata sets, primarily due to excess random accesses. As metadata sets are queried based on specific attributes, such as the source IP address for the geo-location information, shuffling the log records to the bolts will result in random accesses when querying the metadata sets. To overcome this overhead, we group the incoming record stream according to the source IP address, which is used for acquiring the extra contextual information. This results in higher data locality, as each thread will process only a specific range of IP addresses, of which the corresponding metadata information will usually reside in the OS file cache, or even in the CPU cache.

We use Elasticsearch [4] as an external datastore. However, other JSON-enabled datastores, such as MongoDB, can be used as well. We chose Elasticsearch because it is easy scalable and index data in near real-time, giving us a possibility to analyze the more recent information. Moreover, Elasticsearch does not require a fixed datastore schema. That allows us to enrich the data with new contextual information without retroactive updates.

Table 1: Amplification Constants

Protocol	Port	Size (bytes)	BAF
QOTD	17	9.0	140.3
CHARGEN	19	9.0	358.8
DNS	53	78.1	28.7
NTP	123	16.1	556.9
NETBIOS	137	58.0	3.8
SNMP	161	46.0	6.3
RIPv1	520	32.0	131.2[1]
MSSQL	1434	9.0	25[2]
SSDP	1900	100.9	30.8
SIP	5060, 5061	420.4	60 [42]

3.3 Visualization Platform

There are many commercial and open source platforms for data visualization (e.g., Kibana [7], Grafana [5]). Such platforms allow developers to build different types of graphs and analyze data stored in indexed datastores like Elasticsearch [4]. Unfortunately, their power is limited if non-trivial data analysis with inference is required, which is typically the case in cyber security analytics.

To perform visualization and data analysis that are tailored to our specific task, we developed a custom, offline visualization platform using `R` [14] and `Shiny` [13]. This platform can be used by ISPs to design traffic filters and network access rules, and retroactively assess their potential influence.

The visualization platform runs an R webserver that fetches data from Elasticsearch using the `Rcurl` library. Currently, we query all relevant records from Elasticsearch and process them on the webserver side. This requires the host, which runs the R server, to have the capacity to cope with the required processing. We plan to use the rich functionality of Elasticsearch for metric and aggregation calculations, in order to offload some of the computations from the R server. The client can be any modern web browser that renders visualizations produced by the `Shiny` application, and provides means for user interactions with the rendered components.

4 Dataset

In order to visualize the potential gain of the designed mitigation strategies, we need to take into account several factors. First, the size of the incoming request packets varies for different vulnerable protocols. Second, every vulnerable protocol has its own amplification factor. Third, the number of vulnerable servers for every protocol is different in an ISP network. Having this information, it is possible to assess the losses of traffic in an ISP network and evaluate potential gains, as a result of applying mitigation rules. Our visualization tool provides an appropriate interface to specify the corresponding values.

To evaluate the size of request packets for every protocol, we used one day of honeypot data and measured the average payload length of incoming packets for every protocol type. Table 1 reports the corresponding values. Bandwidth Amplification Factor (BAF) is the ratio of the size of the UDP response to the size of the UDP request [40]. We note that amplification factor can vary for different vulnerable servers, even for the same protocol. For instance, a response to a `NTP-monlist` command can return a list of up

[1]`https://www.us-cert.gov/ncas/alerts/TA14-017A`

[2]`https://blogs.akamai.com/2015/02/plxsert-warns-of-ms-sql-reflection-attacks.html`

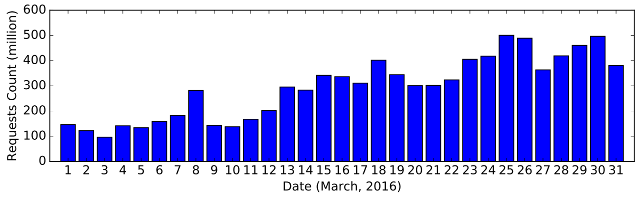


Figure 2: Amount of requests received by our modified AmpPot

to 600 client IP addresses. Obviously, if the number of clients is less, the response will be shorter. As such, the amplification factor will be lower for this server. Given that there are many vulnerable servers in a network and their responses size vary in time, it is difficult to infer precise value for this parameter. For the sake of simplicity, in this paper we use the average values as have been reported by monitoring big ISP networks [40]. Table 1 reports the amplification factors used in this work.

In our tool, the number of vulnerable servers for each protocol is parametrized. Such information may be obtained by the ISP scanning periodically its network in order to identify services vulnerable to the amplification attacks or by using the outcomes of the Censys project [22]. As we currently do not have this information, we set this parameter to 1 for every protocol. However, to estimate roughly the amount of vulnerable services in a network, the interested reader is referred to [40].

To obtain real numbers for amplification attacks, we deployed our amplification honeypot in our organization's network. For the visualization, we selected a continuous one-month dataset from March 1, 2016 to March 31, 2016. Figure 2 shows the number of requests received per day. As can be seen, the number of incoming requests has an increasing trend. On average, we received around 290 million requests per day. Thus, our platform deals with a large amount of data every day, and our choice to rely on the architecture, which easily scales horizontally, is justified.

5 Visualisation Design

In what follows, we describe the interface of our tool, and show how it can be used to develop DRDoS attacks mitigation strategies. We propose a new graphical metaphor called *VizFilt*, which enables the user to develop filters representing mitigation strategies and evaluate their effectiveness on historical data before online deployment.

5.1 Definitions

Actionable attributes ($a_n \in A$) are features that can be directly extracted from each request packet, such as the source IP address, the source/destination port numbers, and the TTL value. In contrast, *informational attributes* refer to the metadata features that are typically appended after contextual enrichment, such as geolocation information, reverse DNS-lookup, and as a result, they cannot be part of filtering rules that are applied at the network level.

A *filter* $F = R_1 \vee R_2 \vee \cdots \vee R_n = \bigvee_{i=1}^{n} R_i$ is a logical disjunction of n boolean rules. A *rule* R_i is a conjunction of positive or negative disjunctions of specific attribute levels, defined by:

$$R_i = \bigwedge_{a_k \in A_{R_i}} LS_k = \bigwedge_{a_k \in A_{R_i}} (n_k \bigvee_j l_k^j),$$

where $A_{R_i} \subseteq A$ is a subset of actionable attributes used in rule R_i, $\bigvee$ and $\bigwedge$ represent boolean algebra operators *OR* and *AND*, and LS_k is a logical statement about the k^{th} actionable attribute $a_k \in A_{R_i}$. LS_k is formed of two parts: a disjunction of level values with l_k^i the i^{th} level of the attribute a_k, and a parameter $n_k \in \{1, \neg\}$ which allows negating (NOT) the disjunction when set to $\neg$. The user can design a rule specifying the levels l_k^i and the parameters n_k.

A filter is composed of as many rules as desired. For example:

$$\begin{array}{rcrcl} F(p) = & & R_1 \vee R_2 & & \\ = & & (dst\_port_p & == & [123\ OR\ 17\ OR\ 53] \\ & AND & ip24_p & == & NOT[128.232.83.0/24]) \\ OR & & (src\_addr_p & == & [123.235.8.9] \\ & AND & ttl_p & == & NOT[90\ OR\ 200] \\ & AND & ip24_p & == & [24.5.42.0/24]) \end{array}$$

where in the first rule R_1: $a_1 = dst\_port$, $n_1 = 1$, $l_1^1 = 123$, $l_1^2 = 17$, $l_1^3 = 53$, $a_2 = ip24$, $l_2^1 = 128.232.83.0/24$ and $n_2 = \neg$ for instance.

An attribute of a packet, which does not appear in a rule, is ignored meaning this attribute can take any level without affecting the outcome of the rule. If the attributes of a packet p match the logical definition of F, then $F(p) = 1(TRUE)$ and p is blocked.

Given the fact that each attribute appears only once in a packet, and its levels are mutually exclusive, we can prove that any possible packet can be filtered with any such rule, and the filter being a disjunction of rules allows blocking any set of packets differing or not by some or all of their actionable attributes.

A blocked request packet p decreases the total bandwidth in an ISP network by a quantity $b(p) = \#amplifiers_{Port_p} \times Size_{Port_p} \times BAF_{Port_p}$, where $\#amplifiers_{Port_p}$ is the amount of amplifiers in the network for the protocol identified by the port $Port_p$, $Size_{Port_p}$ is the average size of the request packet payload, BAF_{Port_p} is the BAF (see Section 4). Summing the quantities $b(p)$ for all packets p blocked by F on a given time window TW, gives the bandwidth recovery $B_r(F, TW)$ enabled by applying F:

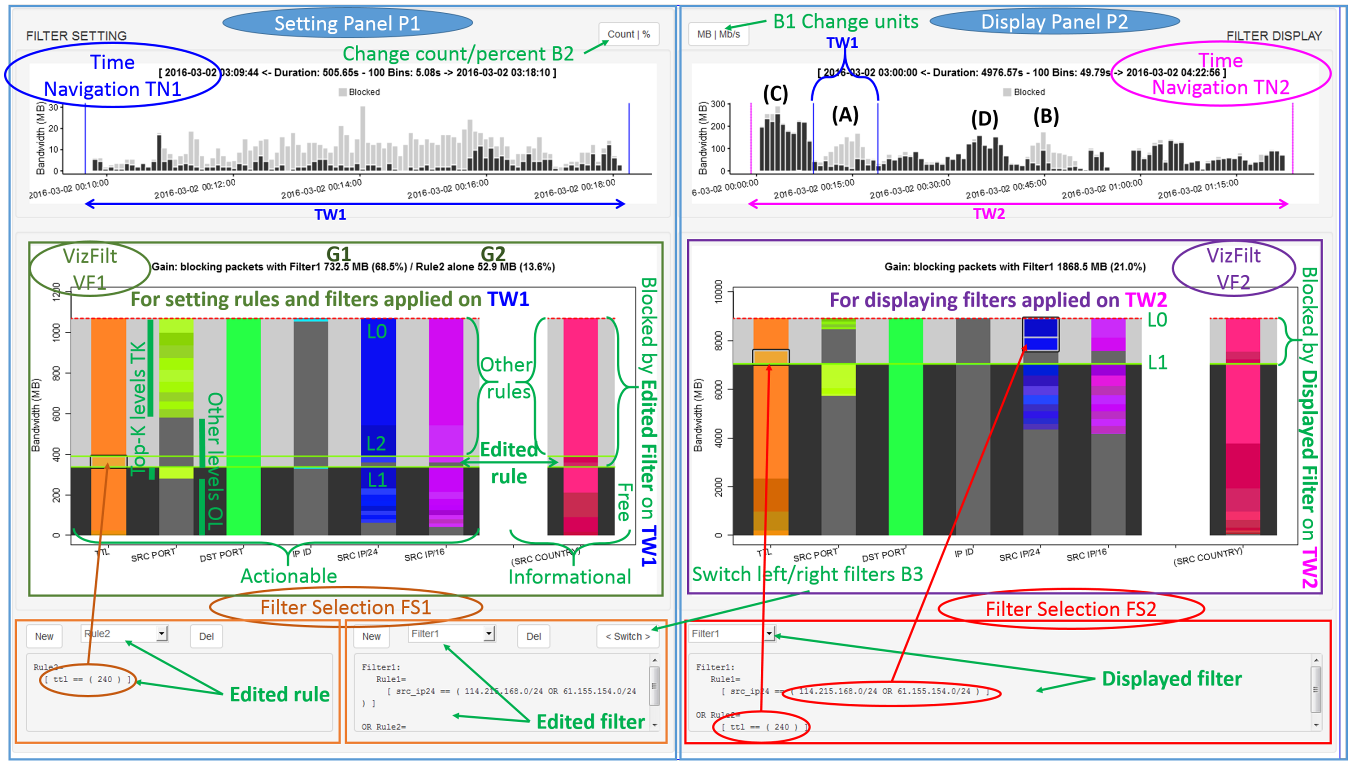
$$B_r(F, TW) = \sum_{p \in TW} F(p) b(p)$$

Finally, the *gain* in bandwidth is computed as $g(F, TW) = B_r(F, TW)/B_{actual}(TW)$, where $B_{actual}(TW) = \sum_{p \in TW} b(p)$ is the actual bandwidth occupied during the time window under focus.

5.2 Anticipated User Requirements

The target user of our system is an analyst working for an ISP trying to estimate the amount of bandwidth wasted due to the participation of ISP's clients in amplification attacks. Thus, the system should facilitate the following processes to the user: (R1) creation of rules that maximize the gain due to the blocked traffic; (R2) combination of several rules into filters; (R3) comparison of different filters; (R4) assessment of gains on different time ranges; (R5) identification of regular traffic and (R6) interactivity.

These requirements stem from the following peculiarities of existing mitigation systems. (R1) The ISP cannot simply block all the traffic coming to the vulnerable services because they are also used for benign purposes. For instance, benign clients can use DNS resolvers within ISP's network to resolve DNS names, although the same resolvers can be also used for amplification attacks. Therefore, developed rules should be very specific blocking only spoofed traffic from the victim. At the same time, these fine-grained rules should supply the best gain. (R2) As uploading rules to network facilities is a costly operation, it should occur rarely. Therefore, the system should provide possibilities to combine several rules into a filter, i.e. a more coarse-grained structure. (R3) The user should be able to define several filters and have the possibility to evaluate which one is better depending on traffic conditions. (R4) Various filters can have different effectiveness on diverse time ranges, thus, the user interface should facilitate the process of evaluating the gain on different time windows. (R5&R6) The user does not know in advance which packets must be blocked. The design of filters's rules is an iterative and manual process. It strongly relies on feedback observing the positive result of decreasing the bandwidth and the negative side effect of affecting the rest of the (regular) traffic. (R5) Identifying the regular traffic strongly relies on user knowledge and


FILTER SETTING
Setting Panel P1
Count | %
Change count/percent B2
Time Navigation TN1
[2016-03-02 03:09:44 <- Duration: 505.65s - 100 Bins: 5.08s -> 2016-03-02 03:18:10]
Blocked
Bandwidth (MB)
TW1
MB | Mb/s
B1 Change units
Display Panel P2
FILTER DISPLAY
[2016-03-02 03:00:00 <- Duration: 4976.57s - 100 Bins: 49.79s -> 2016-03-02 04:22:56]
Time Navigation TN2
(C)
(A)
(D)
(B)
TW2
VizFilt VF1
G1
G2
Gain: blocking packets with Filter1 732.5 MB (68.5%) / Rule2 alone 52.9 MB (13.6%)
For setting rules and filters applied on TW1
Top-K levels TK
Other levels OL
L0
L2
L1
Other rules
Edited rule
Free
Blocked by Edited Filter on TW1
TTL
SRC PORT
DST PORT
IP ID
SRC IP/24
SRC IP/16
(SRC COUNTRY)
Actionable
Informational
VizFilt VF2
Gain: blocking packets with Filter1 1868.5 MB (21.0%)
For displaying filters applied on TW2
Blocked by Displayed Filter on TW2
Switch left/right filters B3
Filter Selection FS1
Filter Selection FS2
New
Rule2
Del
Edited rule
Filter1
< Switch >
Edited filter
Displayed filter

```
Rule2=
  [ ttl == ( 240 ) ]
```

```
Filter1:
   Rule1=
      [ src_ip24 == ( 114.215.168.0/24 OR 61.155.154.0/24
) ]

OR Rule2=
```

```
Filter1:
   Rule1=
      [ src_ip24 == ( 114.215.168.0/24 OR 61.155.154.0/24 ) ]

OR Rule2=
   [ ttl == ( 240 ) ]
```

Figure 3: Overview of the interface structure and illustration of Use Case Scenario 2.

can be supported by both actionable and informational attributes. Our system should let the user to see the effects on both kinds of attributes, while applying filters on actionable ones. (R6) In order to ease the building process, the system should be interactive, displaying the effects of applying a filter instantly after each user action.

Finally, blocking a packet typically comes at some cost. Ideally, the system should provide information about the cost of applying a filter. However, this cost is very difficult to estimate because it involves many factors. Therefore, our system currently relies on domain knowledge of the user to estimate this cost.

5.3 Interface

The interface of our platform is shown in Figure 3 (in the sequel, all symbols refer to this figure and to the user requirements in Section 5.2). The interface is made of two essentially identical panels (P1, P2), each made of three components: the *Time Navigation* component (TN1, TN2) at the top (R4), the new *VizFilt* component (VF1, VF2) at the center (R1, R5), and the *Filter Selection* component (FS1, FS2) at the bottom (R2). The left *Setting* panel (P1) enables filters edition and evaluation, while the right *Display* panel (P2) is only used for comparison to P1 (R3). Any action on a graphical object takes effect instantly on any linked one (R6).

Time Navigation (TN) The TN components of both panels are strictly identical in appearance and interactions. They use a time series graphical representation to display the actual and possibly blocked bandwidth or traffic quantity per time units. Detailed characteristics about the time windows are displayed at the top of these components. Horizontal brushing allows the user to select a time window that automatically takes effect on packets displayed in both the TN and VF components of the same panel. The two TN components display both ends of the time windows (TW1, TW2) selected in the other TN component as vertical blue (TW1) and violet (TW2) lines for the TN1 and TN2 components respectively. This enables the comparison between applying the same filter in different time windows or different filters in the same time window.

A button (B1) is used to switch between *Traffic rate* (expressed in Mb/s) and *Bandwidth* (expressed in MB). When bandwidth is displayed (see Figure 4 Left), the graphical metaphor used is a stacked bar chart with equal-width bars which best encodes an amount of bandwidth during the corresponding time intervals. When traffic rate is displayed (see Figure 4 Right), we use a stacked line (area) chart better expressing the instantaneous nature of the displayed quantity. In both cases, we use stacked bars or lines to express that blocked packets are part of a whole set of packets, and we use a color-code to distinguish which items represent blocked (light grey) and free (dark grey) packets. In Figure 4 we see the result of another mode controlled by button B2, which allows switching between top (C-F) and bottom (D-E) position for displaying the blocked packets, to ease the reading of the amount of free and blocked packets respectively, and between count (C-D) and proportion (E-F) for encoding the total height. In case of proportions, units are changed to percent and all bars have equal total height of 100%.

Filter Selection (FS) At the bottom of the interface are the *Filter Selection* components (FS1, FS2) made of a set of buttons and drop-down lists, and a text area displaying a textual description of the filter displayed in FS2, and of the rule and the filter currently edited in FS1. The text areas have scroll-bars when the rule or filter displayed is too long to fit in the assigned space. In FS1, a drop-down list allows selecting the current rule for editing, and another allows the user to select the edited filter. Both drop-down lists have "New" and "Del" buttons to create or delete rules and filters respectively. In FS2, only the drop-down list to select a filter is present, which applies only to the components of P2. A button B3 allows switching the filters between the Setting Panel P1 and Display Panel P2 together with their respective time windows (TW).

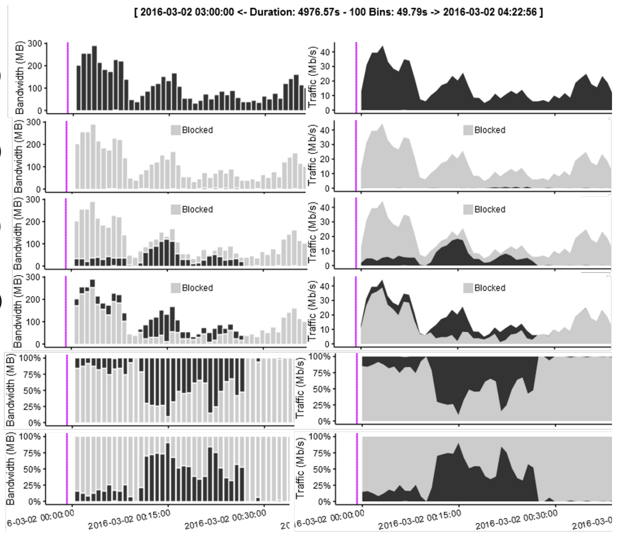


Figure 4: Bandwidth scale is displayed with a stacked bar chart (Left column) and Traffic scale with a stacked line chart (Right column). Blocked packets (light grey) are displayed in terms of count at the top (C) or the bottom (D), or in terms of proportion at the bottom (E) or the top (F) of the stacked bars or lines. (A-B-C) show the result of the Use Case Scenario 1 in same order as in Figures 5 and 6.

VizFilt (VF) - Structure The new visual metaphor called *VizFilt* (VF1, VF2) is central to our design. VF1 allows the user to build a logical rule (Figures 5 and 6) by interacting with the graphical objects based on visual information about the distribution of attributes' levels of all the packets in the time window selected in TN1. We focus on VF1, as VF2 is identical but display-only with no interactive feature except highlights on hovering.

In VizFilt, the bars a separated in two sets: the *actionable* attributes are displayed first from the left, then come the *informational* attributes as a second group of bars clearly separated from the former by an empty space. The name of the attribute encoded by a bar is given on the x-axis. Names of informational attributes are put between parentheses to emphasize their non-actionable nature. The total height of each bar equals the total bandwidth $b(p)$ occupied by the packets $p \in TW$ observed in the selected time window TW of the corresponding panel. As each packet takes a specific level for each of the actionable and informational attributes, the total height is identical for each attribute, however the share of the different levels is different from one attribute to another.

When there are more than a hundred of levels for an attribute, the number of segments in the stacked bar becomes hardly visible and could be even higher than the number of pixels in a column so the user could not distinguish between the segments anyway. As the user is interested in identifying and possibly blocking the packets bearing the *largest* amount of bandwidth, only the tallest segments are of importance. Therefore we consider only the *top-K* levels (TK) of each attribute in the given time window, putting in a supplementary segment all the other levels (OL). The attributes are assigned a specific hue evenly selected on the hue circle in the HSV color space, to maximize the contrast between any two attributes. The last segment (OL) is colored in grey for all the attributes. In order to distinguish the levels within an attribute, the top-K segments are color-coded with small random variations of the hue, saturation and value near the dominant hue of their attribute. This color is set for all the levels in the original dataset to maintain color-codes assigned to levels when changing the time window and related top-k ranking.

VizFilt accounts for packets blocked by a filter, by subsetting the segments of each bar accordingly. As the amount of bandwidth occupied by blocked packets in the time window is the same for all the attributes, we can stack the bars dedicated to blocked packets at the top of the bars in the *blocked-band*, and the ones dedicated to the free packets at the bottom in the *free-band*. The color-code of the segments of the bars in both bands is the same and specific to the levels they represent. We draw a red dashed horizontal line (L0) at the top of the bars to represent the actual bandwidth before filtering, and a green horizontal thick line (L1) between the free-band and the blocked-band that represents the bandwidth after filtering. We fill in the vertical space between the bars in the blocked and free bands, with the same light and dark grey colors used in the TN components to encode blocked and free packets respectively.

When a filter contains more than one rule, the bars of blocked packets are split horizontally with another thick green line (L2) in VF1 only. This line is drawn to separate the packets blocked by the rule currently edited from the other blocked packets. As a filter is a disjunction of rules, it is possible that some packets are blocked by several rules at a time. Therefore, we first apply all the other rules of the filter to get the amount of bandwidth blocked by the filter ignoring the edited rule. Then we apply the edited rule on the remaining packets not yet blocked, to get the minimum amount the edited rule effectively contributes to block by itself, so its added-value to the edited filter. The added-value of the edited filter including all its rules, is given by $g(F,TW)$ (G1). The specific added-value to F of the single edited rule R_e is defined as (G2):

$$B_r(R_e,TW) := B_r(F,TW) - B_r(F_{-e},TW)$$

with $F_{-e} = \bigvee_{i \neq e} R_i$ and $R_e = F_e$. Both these gains (G1,G2) are displayed at the top of the VF1 component, but only is the gain G1 of the displayed filter at the top of VF2. Applying the edited rule after all other rules in the filter allows building a filter incrementally with no overlap between the rules, at least on the packets in the current time window TW1. Note that this distinction (L2) between edited and other rules of a filter does not exist in VF2 which shows the result of applying all the rules of the displayed filter (FS2) at once.

VizFilt (VF) - Interactions Hovering over any segment of the VizFilt component in any panel, highlights its border in white, and displays a tool tip text giving the name of the level and the amount of bandwidth occupied by all the packets with this level in the time window. For OL segments, we display ”$\_N\_OTHERS\_$” where N is the number of levels they aggregate. We also highlight in light yellow the segments of other bars that contain the same packets, in proportion of their occupied bandwidth (Figure 5). These hints help the user connect the packets' information to external knowledge and decide about blocking them or not.

In the VF1 component, the user can create or remove levels in a rule by clicking on the segments of actionable bars. Only the segments of packets blocked by the edited rule and the ones of the free packets react to clicks in that way. As a click on a segment changes the edited rule, the updated filter is applied immediately to the packets in the time window TW1, and the results appear immediately in VF1 and TN1 to reflect the current state of the edited rule and filter. We detail below how clicks on segments affect the edited rule:

- A **click on a *free/blocked* segment whose level l is not in the edited rule** R_e, adds this level to the rule (Figures 5 A→ B and 6 A→ B) *without a prefix/with a NOT prefix*, moves this segment to the *blocked/free* band and makes its border a *thick black plain border (tbpb)/thick black dashed border and cross overlaid (tbdbco)* to states it is now part of the rule in a *positive/negative* disjunction, telling *”l must be blocked if conditions on other features in R_e are met”/”l must not be blocked whatever the other conditions in R_e”.*

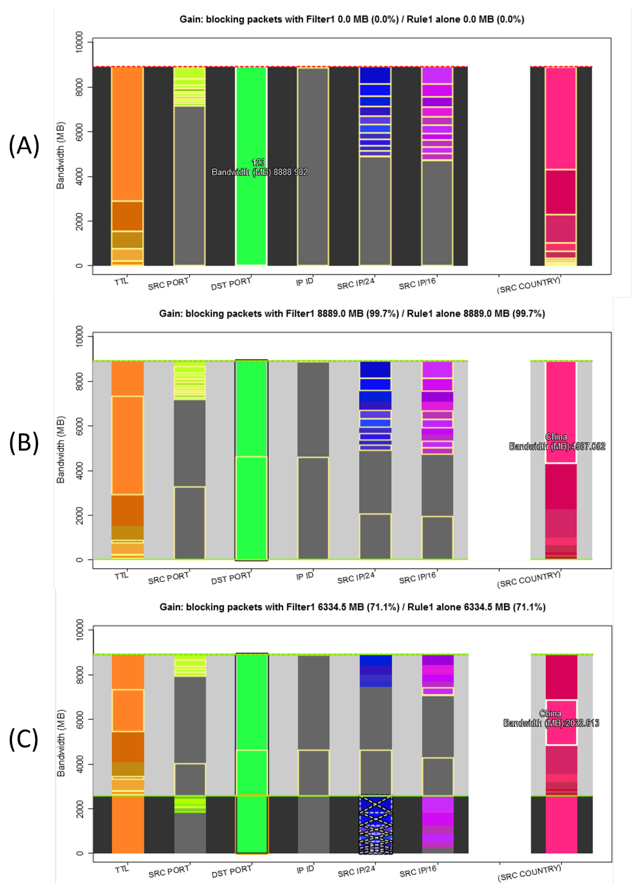
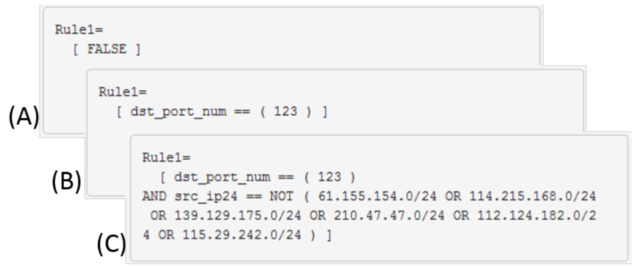


Figure 5: VF1 in Use Case Scenario 1: Building a Filter.

- A **click on a** ***free/blocked*** **segment whose attribute is already in the rule** ***with a NOT prefix/without a prefix*** (Figures 5B→ C and 6 B→ C), *removes/adds* the NOT prefix, moves all the rule's segments of this attribute to the *blocked /free* band, and add a *tbpb/tbdbco* to all of them.

- A **double-click on a segment** removes its level from the edited rule if ever it was part of it, removes its border and possibly moves it to the free or blocked area depending on the result of applying the updated filter.

In all theses cases, all other segments not clicked directly are split between the newly blocked or freed packets, and the other packets, and this new blocked or free segment moves to the blocked or free band respectively.



```
Rule1=
  [ FALSE ]
```

```
Rule1=
  [ dst_port_num == ( 123 ) ]
```

```
Rule1=
  [ dst_port_num == ( 123 )
AND src_ip24 == NOT ( 61.155.154.0/24 OR 114.215.168.0/24
 OR 139.129.175.0/24 OR 210.47.47.0/24 OR 112.124.182.0/2
4 OR 115.29.242.0/24 ) ]
```

Figure 6: FS1 in Use Case 1: the textual rule displayed in the text area correspond to the cases of Figures 4 A-B-C and 5 A-B-C.

When the time window changes, the packets to which applies the filter may differ from the ones used to build the filter. We keep visible the levels taking part of the edited rule or filter even if no packet is blocked, by letting them with a thick and possibly dashed border and overlaid cross, but with an *orange color* instead (Figure 5 C). It reminds the user that these levels exist in the corresponding packets but that not all the conditions of the filter are met for these packets to be blocked.

6 Anticipated Use Case Scenarios

We propose two use case scenarios of interest to an ISP to illustrate our system in a realistic setting based on the data described in Section 4.

6.1 Use Case 1: Building a Filter

In our first use case scenario, the user aims to use VizFilt to build a filter. The user builds the filter containing a single rule, resulting in the views A-B-C of the figures 4 (TN1), 5 (VF1) and 6 (FS1). In the initial VizFilt metaphor (5A) with an empty rule, the user selects the destination port 123, by clicking the corresponding green bar (*DST_PORT*), resulting to the creation of the corresponding rule (6B). At the same time, the full bar has moved to the blocked-band (top of 5B-C) corresponding to the blocked packets in TW1, allowing her to see that by blocking this port many packets from *SRC_COUNTRY China* (pink color) will be blocked as well, which might not be desirable. The user decides to prevent this by clicking the blue blocked segments whose *SRC_IP*24 correspond to *China* has visible by the light yellow overlay (5B). This will result to the creation of the rule shown in 6C, in which the *SRC_IP*24 that should not be blocked, have been added to a boolean NOT expression. These segments are now in the free-band (bottom) with a cross overlaid. Part of the previously blocked packets in the *DST_PORT* attribute are now back into the free-band too as their *SRC_IP*24 attribute matches with the newly unblocked levels, still they appear as a green bar with an orange border highlighting that the 123 level belongs to the rule but cannot block these packets.

6.2 Use Case 2: Comparing Different Time Windows

In our second scenario, the user aims to explore if the design of a filter in some specific time range is effective in other time ranges (Figure 3). After having detected a spike in the full time range in the TN1 (not shown), the user brushes a small time range TW1 around the spike to zoom in (A in TW2). Then, the user builds a filter with two rules (FS1), that gives good results on TW1 (G1, G2). In order to evaluate this filter on the full time window, the user can select the same filter in the right panel (FS2), and verify how it applies on the larger time range TW2. By doing that, the user can perceive that the filter designed in TW1, is also efficient to block some spikes of TW2 (B), but not to block all of them (C, D) so she can decide to set up a new rule to handle these cases.

7 Conclusions

DRDoS attacks represent a growing threat. During the last couple of years, the attacks using this technique achieved an accumulation of 500 Gbit/s of undesired traffic. Due to the distributed nature, the traffic from amplifiers may remain undetected, however, it consumes a significant portion of the network bandwidth, causing unnecessary money expenditures. In this paper, we proposed a method and a tool that allow the ISP to evaluate the harm caused by innocent participation of its clients in the amplification attacks. Our developed platform collects the data required to get actionable knowledge while our custom visualization tool allows an operator to develop mitigation rules, simulate their appliance, and evaluate their effectiveness on historical data. We are currently collaborating with an ISP to evaluate and improve our visualization solution.

References

[1] Apache Kafka: A High-Throughput Distributed Messaging System. http://kafka.apache.org/.

[2] Apache Metron: Real-Time Big Data Security. http://metron.incubator.apache.org.

[3] Apache Storm. http://storm.apache.org/.

[4] Elasticsearch: Search & Analyze Data in Real Time. https://www.elastic.co/products/elasticsearch.

[5] Grafana 3.1.0 released. http://grafana.org/blog/2016/07/12/grafana-3-1-released.html.

[6] IBM Security Intelligence with Big Data. http://www-03.ibm.com/security/solution/intelligence-big-data/.

[7] Kibana: Explore & Visualize Your Data. https://www.elastic.co/products/kibana.

[8] LogRhythm Security Analytics. https://logrhythm.com/products/security-analytics/.

[9] OpenSOC: Big Data Security Analytics Framework. http://opensoc.github.io.

[10] Pravail Security Analytics. https://www.pravail.com.

[11] RSA Security Analytics. http://www.emc.com/collateral/data-sheet/security-analytics-overview-ds.pdf.

[12] Securonix Security Analytics Platform. http://www.securonix.com/security-intelligence/.

[13] Version 0.13.2 released. http://shiny.rstudio.com/.

[14] Version 3.2.3 released. https://www.r-project.org/.

[15] Arbor Networks. DDoS Attacks in the Gaming Industry. https://resources.arbornetworks.com/h/i/153941358-ddos-attacks-in-the-gaming-industry/, October 19 2015.

[16] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita. An Empirical Evaluation of Information Metrics for Low-rate and High-rate DDoS Attack Detection. *Pattern Recognition Letters*, 51:1–7, 2015.

[17] CERT Coordination Center. Results of the Distributed-systems Intruder Tools Workshop, year = 1999. *Software Engineering Institute*.

[18] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks. In *ACM IMC*, 2014.

[19] D. Dittrich. The DoS Project's "trinoo" Distributed Denial of Service Attack Tool. https://staff.washington.edu/dittrich/misc/trinoo.analysis.txt/, October 21 1999.

[20] D. Dittrich. The "stacheldraht" distributed denial of service attack tool. https://staff.washington.edu/dittrich/misc/stacheldraht.analysis/, December 31 1999.

[21] D. Dittrich. The "Tribe Flood Network" Distributed Denial of Service Attack Tool. https://staff.washington.edu/dittrich/misc/tfn.analysis/, 1999.

[22] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A Search Engine Backed by Internet-wide Scanning. In *ACM CCS*, 2015.

[23] Z. Durumeric, M. Bailey, and J. A. Halderman. An Internet-Wide View of Internet-Wide Scanning. In *USENIX Security*, 2014.

[24] R. F. Erbacher, K. L. Walker, and D. A. Frincke. Intrusion and Misuse Detection in Large-scale Systems. *IEEE Computer Graphics and Applications*, 22(1):38–47, 2002.

[25] K. Gangavarapu, V. Babji, T. Meiner, A. I. Su, and B. M. Good. Branch: An Interactive, Web-based Tool for Testing Hypotheses and Developing Predictive Models. *Bioinformatics*, 32(13):2072–2074, 2016.

[26] L. Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, 33(4):12–17, 2000.

[27] L. Girardin. An Eye on Network Intruder-Administrator Shootouts. In *Workshop on Intrusion Detection and Network Monitoring*, 1999.

[28] P. Hick, E. Aben, K. Claffy, and J. Polterock. The CAIDA DDoS Attack 2007 Dataset.

[29] J. Huo and W. B. Cowan. KMVQL: A Graphical User Interface for Boolean Query Specification and Query Result Visualization. In *IEEE VIS*, 2003.

[30] M. Karami and D. McCoy. Understanding the Emerging Threat of DDoS-as-a-Service. In *USENIX LEET*, 2013.

[31] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow. AmpPot: Monitoring and Defending Against Amplification DDoS Attacks. In *RAID*, 2015.

[32] D. Kumar, G. Rao, M. K. Singh, and G. Satyanarayana. A Survey of Defense Mechanisms countering DDoS Attacks in the Network. *Intl. Journal of Advanced Research in Computer and Communication Engineering*, 2:2599–2606, July 2013.

[33] C. P. Lee, J. Trost, N. Gibbs, R. Beyah, and J. A. Copeland. Visual Firewall: Real-time Network Security Monitor. In *ACM VizSEC*, 2005.

[34] Y. Lee and Y. Lee. Toward Scalable Internet Traffic Measurement and Analysis with Hadoop. *ACM SIGCOMM CCR*, 43(1):5–13, 2013.

[35] J. McPherson, K.-L. Ma, P. Krystosk, T. Bartoletti, and M. Christensen. PortVis: A Tool for Port-based Detection of Security Events. In *ACM VizSEC/DMSEC*, 2004.

[36] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM CCR*, 34(2):39–53, 2004.

[37] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[38] N. Murray, N. W. Paton, C. A. Goble, and J. Bryce. Kaleidoquery – A Flow-based Visual Language and its Evaluation. *J. Vis. Lang. Comput.*, 11(2):151–189, 2000.

[39] J. Pearlman and P. Rheingans. Visualizing Network Security Events Using Compound Glyphs from a Service-oriented Perspective. In *IEEE VizSEC*, 2007.

[40] C. Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *NDSS*, 2014.

[41] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt. Amplification and DRDoS Attack Defense – A Survey and New Perspectives. *arXiv preprint arXiv:1505.07892*, 2015.

[42] R. Shankesi, M. AlTurki, R. Sasse, C. A. Gunter, and J. Meseguer. Model-checking DoS Amplification for VoIP Session Initiation. In *ESORICS*, 2009.

[43] H. Shiravi, A. Shiravi, and A. A. Ghorbani. A Survey of Visualization Systems for Network Security. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1313–1329, Aug 2012.

[44] S. T. Teoh, K. L. Ma, S. F. Wu, and X. Zhao. Case Study: Interactive Visualization for Internet Security. In *IEEE VIS*, 2002.

[45] S. van den Elzen and J. J. van Wijk. BaobabView: Interactive Construction and Analysis of Decision Trees. In *VAST*, 2011.

[46] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju. VisFlow-Connect: Netflow Visualizations of Link Relationships for Security Situational Awareness. In *ACM VizSEC/DMSEC*, 2004.