J Netw Syst Manage (2017) 25:735–758
DOI 10.1007/s10922-017-9421-4

Flow-Based Web Application Brute-Force Attack and Compromise Detection

Rick Hofstede[1] · Mattijs Jonker[1] · Anna Sperotto[1] · Aiko Pras[1]

Received: 12 June 2017 / Revised: 8 August 2017 / Accepted: 10 August 2017 /
Published online: 18 August 2017

Abstract In the early days of network and service management, researchers paid much attention to the design of management frameworks and protocols. Since then the focus of research has shifted from the development of management technologies towards the analysis of management data. From the five FCAPS areas, security of networks and services has become a key challenge. For example, brute-force attacks against Web applications, and compromises resulting thereof, are widespread. Talks with several Top-10 Web hosting companies in the Netherlands reflect that detection of these attacks is often done based on log file analysis on servers, or by deploying host-based intrusion detection systems (IDSs) and firewalls. However, such host-based solutions have several problems. In this paper we therefore investigate the feasibility of a network-based monitoring approach, which detects brute-force attacks against and compromises of Web applications, even in encrypted environments. Our approach is based on per-connection histograms of packet payload sizes in flow data that are exported using IPFIX. We validate our approach using datasets collected in the production network of a large Web hoster in the Netherlands.

This paper is based on Chapter 5 of the first author's Ph.D. thesis. The thesis was defended on June 29, 2016; the title of the thesis is "Flow-based Compromise Detection".

✉ Aiko Pras
a.pras@utwente.nl

Rick Hofstede
r.j.hofstede@alumnus.utwente.nl

Mattijs Jonker
m.jonker@utwente.nl

Anna Sperotto
a.sperotto@utwente.nl

[1] University of Twente, Enschede, The Netherlands

Keywords Flow monitoring · IPFIX · Intrusion detection · Compromise detection · Web

1 Introduction

In the early nineteen-nineties, thus at the time JNSM was established, the research community was focussing on the development of management frameworks, architectures, and on the design of management protocols. At that time, JNSM, but also conferences such as IM, were platforms to discuss the protocol wars between the ISO–OSI (and later ITU) world at one side, and the Internet (IETF) world at the other side. Researchers seemed to believe that, once management technologies were in place, solving management challenges would become easy.

Now, 25 years later, we have learned that the key to solve complex management problems is not so much in the design of new management technologies, but rather in the clever analysis of available management data. Such data may not only be retrieved from the MIBs that are found within management agents, but can also be directly obtained from monitoring traffic within networks. An interesting network monitoring approach that has become popular within the last decade, is the analysis of flows that are exported using protocols such as IP flow information export (IPFIX) [6]. In that context, a flow is defined as "a set of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties." Individual packets to and from, e.g., Web servers, can be aggregated into flows by a flow exporter/probe, and sent to a flow collector for storage and analysis, as shown in Fig. 1.

Since the early days of research, the management problems that needed to be solved focussed on five functional areas: fault, configuration, accounting, performance and security (FCAPS). Whereas initially *performance* and *fault* management attracted quite some attention, it is clear that nowadays *security* has become a key problem. In this paper we will therefore demonstrate, based on the example of securing Web applications, how data exported using IPFIX can be analyzed in a clever way to solve current security management problems.

Web applications are widespread and their operation is crucial in our daily lives. Examples of popular Web applications are content management systems (CMSs), such as WordPress, Joomla, and Drupal. The popularity of CMSs, the aforementioned three in particular, is also underlined by numbers: one-third of all Web sites on the Internet are built using these CMSs.[1] The widespread use of these CMSs also comes with a risk: the fact that anybody can use them, even people with limited technical skills that are unaware of security threats and measures, leads to outdated and vulnerable CMSs, and reliance on weak administrator passwords [7]. As such, CMSs end up being a prime attack target [10, 19, 28] and the number of attacks is increasing every day. The security company Sucuri visualizes failed login attempts on WordPress instances behind their protection services, which shows an increase up to a factor eight,

[1] http://w3techs.com/technologies/overview/content_management/all

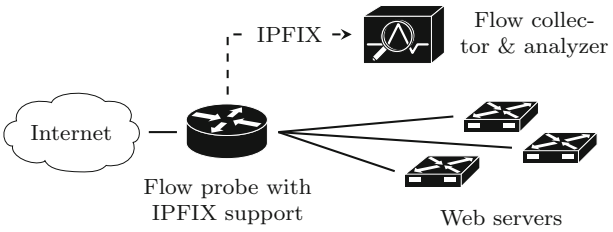


Fig. 1 Flow monitoring for Web servers using IPFIX

over a 6-month period [23]. Brute-force attacks on Web applications are a major security threat for multiple reasons [19]. First, these attacks result in an increased load on the underlying infrastructure. Second, following a compromise, malicious scripts can be installed, such as Web shells,[2] [12, 18]. Third, Web applications can be misused for a wide range of illegal activities: distribution of pirated content and malware [5], SPAM campaigns, participation in botnets and Distributed Denial of Service (DDoS) attacks, etc. In such cases the entire IP-space owned by the hosting company may get black-listed, thus a security mistake made by a single customer potentially impacts all other customers of the hosting company.

Detecting attacks against Web applications can be done in several ways. From talks with several Top-10 Web hosting companies in the Netherlands we have learned that the detection of attacks in a host-based fashion (thus on the machine that runs the Web applications) is by far the most popular approach. Such protection can be realised, for example, by using CAPTCHA or IP-based authentication blockers. However, such blockers must be implemented and maintained by the customers, who generally have limited technical skills. A better approach may therefore be to apply central authentication monitors that analyze all log files on Web servers on-the-fly and block attackers by IP address after a certain number of failed authentication attempts. These monitors come with so-called Web panels—administration interfaces for Web hosting products. The fact that all three major Web panels, namely cPanel, Plesk and DirectAdmin [1], provide this functionality, indicates that this specific form of attack detection is used by many Web hosting companies. Unfortunately this approach can only work if the Web hosting company has access to the complete set of Web server logs; in cases where the hosting company provides Virtual Machines (VMs) to its customers, this approach is likely impossible.

To overcome the problems of host-based detection, this paper discusses the feasibility of a network-based approach, in which only a single sensor needs to be deployed at a strategic observation point. A network-based approach can be implemented in two ways: we either take a specific intrusion detection system, such as Snort or Bro, or we take a more generic approach that is based on established monitoring technologies/protocols, such as IPFIX, for which the collected data can also be used for other purposes than intrusion detection. This papers focuses on the analysis of IPFIX data to detect brute force attacks on Web applications, and in particular investigates whether such analysis can also unveil whether the attack has been successful and the system has been compromised.

[2] *Web shells* provide a remote server shell, yielding serious security risks.

Existing works have shown that promising detection results can be achieved by analyzing flow data of brute-force attacks [8, 9, 25–27], although fluctuations in network traffic, such as transmission control protocol (TCP) retransmissions and control information, may result in both false positives and false negatives [14]. To overcome these problems, the novelty proposed by this paper is to analyze flow data that is enhanced with histograms that describe packet payload distributions. These histograms show not only the total size of a specific flow, as is the case for regular flow data, but the entire payload size distribution. This allows us to discriminate TCP control information and retransmissions from other traffic, among others, such that these phenomena do not impair our data analysis. To extract the per-connection histograms from network traffic, we use a flow exporter, or flow probe, which we have equipped with an extension for exporting histograms for every observed flow, to aid in our detection of attacks against Web applications.

Our contributions can be summarized as follows:

1. We present a network-based approach for detecting brute-force attacks and compromises against Web applications, based on IPFIX, that is resilient against attacks on the core hosting infrastructure (Sect. 4).
2. The use of histograms for detection makes our approach resilient against real-world artifacts introduced by TCP, such as retransmissions and control information, even if the network traffic is encrypted (Sect. 3).
3. Our work has been validated using a month-long dataset of a Top-10 Web hosting company within the Netherlands. The dataset consists of flow data and Web server log files for approximately 2500 Web hosting accounts (Sect. 5).

The remainder of this paper is structured as follows. In Sect. 2, we elaborate on various characteristics of brute-force attacks, such as the phases they exhibit, as well as related work on the detection of these attacks. The statistical information we use for intrusion detection is outlined in Sect. 3, followed by an extensive description of our detection approach in Sect. 4. After that, we describe our validation approach and results in Sect. 5 and discuss possible mitigation approaches in Sect. 6. Finally, we draw our conclusions in Sect. 7.

2 Background and Related Work

Previous studies have shown that brute-force attacks typically consist of three phases [11], as shown in Fig. 2. First, in the *scan* phase, attackers perform horizontal scans over a network to find targets, i.e., active daemons on a particular port. This phase features only a very small number of packets per flow; TCP's three-

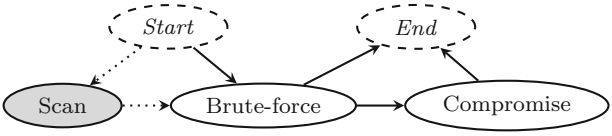


Fig. 2 Dictionary attack phases, from [11]

way handshake is sometimes even stopped prematurely. Second, in the *brute-force* phase, attackers perform the actual attack by trying to authenticate against a daemon or service using *dictionaries*, lists of frequently-used username and password combinations. The *brute-force* phase is generally the longest and most intense attack phase, and features a significantly larger number of packets per flow than the *scan* phase. Third, in case attacks reach the *compromise* phase, targets have been compromised. Attackers may then either actively misuse targets or leave them aside for the time being. Note that not all attack phases need to be visible within a certain portion of network traffic, since attackers may choose to delay execution of attack phases or execute attack phases from different machines to evade detection [13].

The *scan* phase in the context of attacks against Web applications is different in nature from attacks against other applications. This is because Web applications can typically not be scanned using target IP addresses only, as domain names are required to reach Web applications. Web applications are served by *vhost*s, which can be considered as virtual containers on a Web server, one per domain, such that one Web server can serve multiple domains. Mapping IP addresses to *vhost*s is non-trivial and not directly related to the attack itself, which is why we ignore the *scan* phase in the remainder of this work.

The flow-based detection of brute-force attacks in general is not a new area of research. However, related works have so far only focused on attacks against a specific class of protocols, namely those protocols that are *login-restrictive*, such as Secure SHell (SSH). These protocols require successful authentication to advance in the protocol's state machine. As such, they feature an important characteristic that makes attacks against them relatively easy to detect: Performing brute-force attacks on these protocols yields a very typical traffic pattern caused by many failed authentication attempts. This is because repeated authentication attempts are almost identical application-layer actions. We often refer to this kind of traffic as *flat* traffic, since it features traffic flows in the *brute-force* phase that are alike in terms of the number of packets and bytes, and duration. Most works in the context of flow-based intrusion detection rely on the identification of this flat traffic to find brute-force attacks [8, 9, 25–27]. The *compromise* phase is then typically identified by deviations from the *brute-force* phase traffic pattern [27]. Another approach to SSH compromise detection was presented in [11], where attack tool behavior upon compromise is recognized instead of checking for traffic deviations, which was proven to greatly reduce the number of false positives in production deployments.

The hypertext transfer protocol (HTTP) protocol is certainly not *login-restrictive*, making the detection of brute-force attacks over this protocol more challenging. This is even more true when Secure Sockets Layer (SSL) or Trans- port Layer Security (TLS) is used to encrypt sessions in an end-to-end fashion. Several works (e.g., [17]) have targeted the anomaly-based detection of attacks over HTTP(S), but rely on individual packets for doing so. The only flow-based attempt in this direction is described in [24], where the authors extract attack signatures from tools that can be used for performing brute-force attacks against Web applications. The major disadvantage of this signature-based approach is that it has shown to catch legitimate traffic as well, mostly caused by calendar fetchers and Web crawlers. This is because the approach is based on the assumption that brute-force attacks

consist of many flows that are rather small in terms of packets and bytes, which is however also true for the aforementioned type of Web applications. Although our approach also assumes attack traffic in the *brute-force* phase to be flat, we design our detection approach such that attack traffic can be discriminated from benign traffic that is alike in terms of packets, bytes and durations, by means of using histograms for intrusion detection.

3 Histograms for Intrusion Detection

In this section we demonstrate how histograms can be utilized for intrusion detection. We start by covering background information on histograms and motivating their use in Sect. 3.1. Then, in Sect. 3.2, we explain how to compare histograms and form clusters, while in Sect. 3.3, we provide several concrete examples and measurement-based insights into how indicative attack traffic is when represented using histograms.

3.1 Traffic Characteristics

Key to identifying brute-force attacks in flow data is to aggregate similar records into clusters. Ultimately, records describing the same attack, which are assumed to be rather similar in terms of the number of packets and bytes, and duration, should be part of the same cluster. Several works have however shown that relying on packet and byte counters in flow data should be done with care, especially when it comes to identifying *flat* traffic for network security analysis. For example, it has been shown that TCP retransmissions and control information, which affect the packet and byte counters due to timing parameters, cannot be discriminated in flow data [14]. This causes attacks from countries that are far-away from the observation point—above all in terms of geographical distance—to stay under the radar of Intrusion Detection Systems (IDSs). But even if flat traffic is identified properly, its "detection for HTTP(S) was found to be ineffective, because valid AJAX updates common on Web 2.0 tend to produce flat traffic pattern" [8]. This is also confirmed by [24], where it is shown impossible to differentiate traffic of Web crawlers and calendar fetchers from dictionary attack traffic based on packet and byte counters alone. We therefore support the observations in [14] that flow data must be enhanced by additional fields to become a reliable source of information for intrusion detection.

To overcome the described problems with counters in flow data, we need more granular information on individual packets in a flow. Several metrics could be used for this purpose. For example, we have experimented with exporting packet inter-arrival times in histograms, as it was shown that timing information can be used for identifying applications in flow data [20]; Similar application-layer actions, such as login attempts, would feature similar timing characteristics, allowing for the aggregation of attack flows into the same cluster. We have however found two problems with this approach. First, inter-arrival times do not allow TCP control information packets to be discriminated from other packets, yielding this approach

rather susceptible again to the problems described in [14]. Second, reliable measurements can only be done using special clocks for accurate hardware timestamping, which are neither available in our measurement infrastructure, nor in many others. We have therefore found this approach to be too sensitive to measurement errors, impairing subsequent clustering procedures.

Along with the other previously explained problems, the problems of inter-arrival times can be overcome by using per-flow information on individual packet payload sizes. This even allows us to discriminate TCP control information in flow data, as it is often carried in zero-payload packets. Also, we know that Web crawlers, calendar fetchers and dictionary attacks, for example, use different distributions of packets within flows, which allows us to discriminate these applications using the more granular data. Furthermore, since encrypted channel handshakes result in a constant pattern in every connection, the use of histograms allows for clustering similar encrypted channels. For these reasons, we export—per flow—a histogram with packet payload sizes, such that each *sample* in the histogram represents the payload size of one packet in that flow. To do so, we defined an enterprise-specific Information Element (IE) for IPFIX and instrumented our measurement infrastructure with it. Based on the datasets described in Sect. 5.2, we can conclude that our payload histograms extend the size of a flow record (on the wire) by at most 37 bytes in 99% of all flow records.

The use of histograms for intrusion detection in general is not new. An extensive overview is given in [16], where it is explained how to map network traffic features to histograms, cluster these histograms, and classify anomalous traffic patterns based on the created clusters. While the authors provide examples of various common network traffic anomalies, such as port scans, they demonstrate the generic viability of their histogram-based approach, but do not focus on a specific application or extension as we do in this paper for flow-based brute-force attack and compromise detection. Also, we use a pivotal distance metric for clustering that is not covered in [16], for reasons to be discussed in the next subsection.

3.2 Clustering Histograms

Clustering aims at grouping objects with similar characteristics into sets, such that the sets feature a low inter-set similarity (i.e., large distance between objects in different sets) and a high intra-set similarity (i.e., small distance between objects in the same set). When histograms are used for intrusion detection, it is key to cluster histograms that are ‘similar’ with respect to their bins. Alternative to maximizing the similarity of histograms within a cluster is to minimize the distance between histograms. Well-known and frequently-used distance metrics are the *Manhattan* (L1-norm), *Euclidean* (L2-norm) and *Mahalanobis* distances [16, 21]. The main problem with these distance metrics in the context of our work is that they fulfill the *shuffling invariance* property, meaning that the distance between any two histograms does not change when bin values are interchanged. This problem can be observed in Fig. 3, which shows three payload size histograms in a typical HTTP brute-force attack. Intuitively, we would assume the distance between Histograms A and B to be smaller than the distance between Histograms A and C, given that the

difference between 255 and 259 is much smaller than the difference between 255 and 459. This is however not the case for distance metrics that satisfy the *shuffling invariance* property, as is demonstrated for the Euclidean distance:

$$D(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{1}$$

Using (1), we can calculate the distances between histograms in Fig. 3:

$$D(A,B) = \sqrt{|7-7|^2 + |2-1|^2 + |0-1|^2} \tag{2}$$

$$D(A,B) = D(A,C) = \sqrt{2} \tag{3}$$

Note that both distances are equal, even though the histograms differ significantly, especially when they are visualized to scale. Once we translate this result to network traffic, the unsuitability of the (Euclidean) distance metric becomes clear immediately; The difference between Histogram A and B can easily be caused by variability in the TCP header or differences in username and password lengths, for example, while Histogram C shows significantly different traffic. A solution to this 'problem' is provided in [4], where the Minimum Difference of Pair Assignments (MDPA) distance metric is defined. In a nutshell, MDPA aims at *finding the minimum difference of pair assignments between two sets*, where *sets* are histogram bins in our context:

$$D(x,y) = \min_{x,y} \left(\sum_{i,j=0}^{n-1} d(x_i, y_j) \right) \tag{4}$$

Here, $d(x_i, y_j)$ is defined as the arithmetic difference between bin i in histogram X and bin j in histogram Y. Hence, the more similar any two histograms are, the smaller the value D. For the histograms in Fig. 3, the following distances can be obtained: $D(A,B) = 4$ and $D(A,C) = 204$, as demonstrated in "Appendix 1". From these results it becomes clear what the added value of the MDPA distance metric is, compared to commonly-used metrics like Euclidean. The MDPA distance metric does not satisfy the *shuffling invariance* property, but, besides, is similar in nature to the commonly used distance metrics. We therefore rely on this metric in the

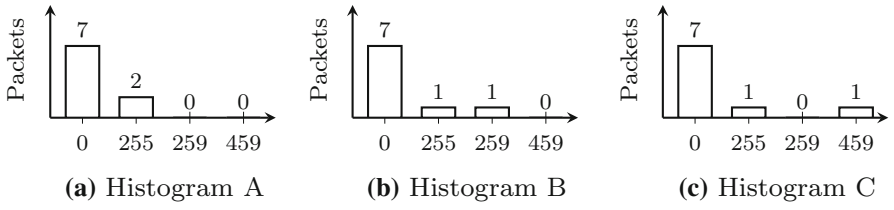


Fig. 3 Payload size histograms in typical HTTP brute-force attack traffic. **a** Histogram A. **b** Histogram B. **c** Histogram C

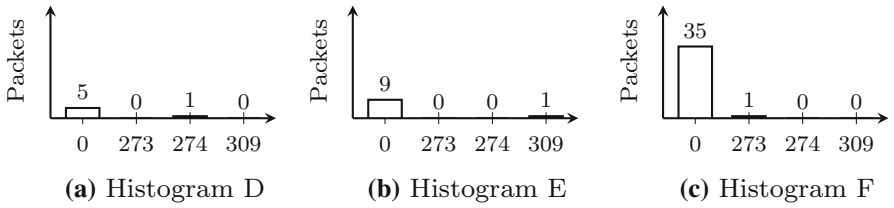


Fig. 4 Payload size histograms in benign HTTP flows. **a** Histogram D. **b** Histogram E. **c** Histogram F

remainder of this work, unless indicated differently. For a detailed MDPA example, we refer the reader to "Appendix 1".

3.3 Measurements

Based on the concept of using histograms for intrusion detection and calculating inter-distances for clustering, we provide in this section several measurement-based insights into how this works in practice.

In Fig. 4, we visualize the payload size histograms of three consecutive flows in a benign client-server interaction with a Web shop. What catches attention are the zero-value bins, which appear to be relatively large in some histograms, such as for Histogram F. Packets accounted in the zero-value bin feature no payload, meaning that they are likely made up of TCP acknowledgements, window updates and other control information. What all histograms in Figs. 3 and 4 have in common are the seemingly minor deviations in the number of bytes. To investigate whether— and if so, until which extent—the password length of authentication attempts affects the size of a flow, we have measured various password lengths and resulting flow sizes in a lab setup. We have done this by generating—per selected password length—100 random passwords that were fed into the *Patator*[3] brute-force attack tool, and captured the resulting traffic. The results, which are shown in Fig. 5 together with their respective standard deviations, indicate that every password character accounts for one byte in the total flow size. Deviations in flow sizes are found to be mainly the result of TCP sometimes dividing the returned Web page over two segments instead of one, resulting in additional acknowledgements. Given that most popular passwords, which are naturally also commonly found in *dictionaries*, typically feature no more than ten characters [2], we conclude that the impact of password lengths on the total flow size is limited.

Another reason for histograms to appear significantly different is when they represent packet payloads in hypertext transfer protocol secure (HTTPS) connections, as opposed to histograms for regular HTTP connections. To investigate how different the resulting histograms are, we have also measured attacks over HTTPS in a lab setup. As shown in Fig. 6, HTTPS connections are bin-wise fundamentally different from their non-encrypted counterparts when compared to the histograms in Figs. 3 and 4: Both the total number of bins and the total flow size are larger, while the number of zero-payload packets is basically the same. The differences can be

[3] Patator v0.7, which can be retrieved from https://github.com/lanjelot/patator

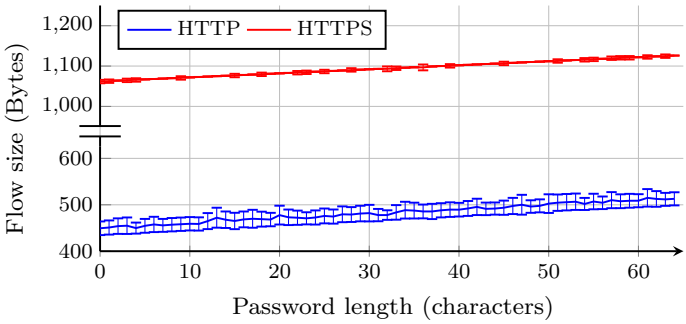


Fig. 5 Impact of password length on total flow size

accounted to the establishment of the encrypted channel, which requires cipher selection, key exchange, etc. However, when considering histograms of benign client-server interactions and brute-force attacks in HTTPS, a characteristic difference in distances similar to that in the case of HTTP traffic can be observed.

The most important take-away from the measurements and histograms presented in this section is that histograms for benign connections are quite different from histograms in typical brute-force attacks, such as those visualized in Fig. 3. We will use this observation in our detection approach, which we explain in the next section, to discriminate *brute-force* phase traffic from other traffic and identify any subsequent compromises.

4 Detection Approach

Our detection approach is based on data exported by a flow exporter using IPFIX, and consists of two phases: *Preselection* and *detection*. The *detection* phase in itself is made up of several steps, as is depicted in Fig. 7. Both phases, and the respective steps they comprise, are explained in the remainder of this section.

As with most flow-based analysis applications, the various analysis steps operate on flow data chunks. Data chunks consist of flow data that has been received in fixed-length time intervals, typically in the order of several minutes. The use of data chunks allows for near-real-time processing of network traffic on the one hand, and demarcates the dataset used per iteration on the other.

4.1 Preselection

The *Preselection* phase serves to make a rough data selection based on a number of criteria, such that the amount of data to be processed in further steps is reduced. Since this has advantages solely in terms of performance, this is an optional step. If used, *Preselection* returns a list of IPv4 and IPv6 address tuples of possible attackers and targets. More formally, we define a *tuple* as a pair of source and destination IP

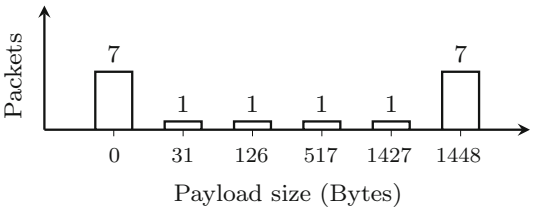


Fig. 6 Payload size histogram of a TLS flow

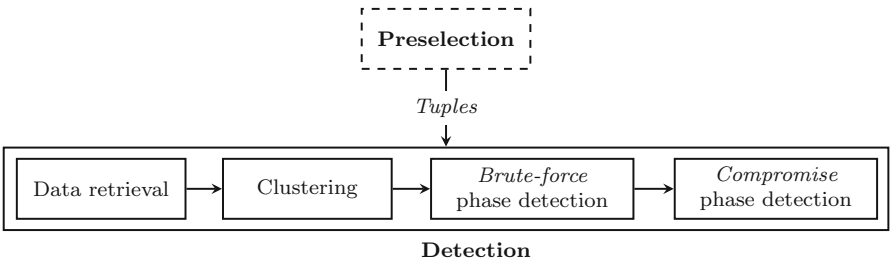


Fig. 7 Conceptual detection approach

addresses, source port number and *vhost*.[4] To qualify for preselection, an attacker must have generated at least N flows towards a target. Note that we refer several times to this number in the remainder of this paper and that the used value for N is explained in Sect. 5.

Since the goal of *Preselection* is data filtering and not detection, exceeding the threshold for tuple qualification is often easy, even for benign applications, such as Web crawlers and calendar fetchers. It is then up to the *detection* phase to classify these cases as benign. Nevertheless, our measurements have shown that *Preselection* can improve the detection process' overall performance by more than a factor 7 in terms of processing time on our validation datasets. This is because the clustering procedure, covered in Sect. 4.3, is by far our most computationally expensive component, so its use should be limited as much as possible by confining the input dataset.

4.2 Data Retrieval

This step retrieves the flow data used within the *detection* phase. In case a *Preselection* was done before, only data that belongs to the preselected tuples within the current data chunk is retrieved, and a full data chunk otherwise. After retrieval, flow data that cannot be part of an attack by definition is filtered out. For example, flow records from attacker to target typically feature at least four packets, because every valid HTTP request consists of the following packets at least:

[4] Many IPFIX flow exporters extract *vhosts*, often referred to as *HTTP hostname*, from HTTP headers. This information is no prerequisite for our detection approach and therefore only used within the *Preselection* phase.

- TCP `SYN`—First packet of three-way handshake.
- TCP `ACK`—Third packet of three-way handshake.
- HTTP `GET`/`POST`—Actual HTTP request.
- TCP `FIN`/`RST`—Connection teardown.

Note that network flows are typically unidirectional in nature, so these packets represent merely the traffic from attacker to target. Since we can only identify TCP flags in flow data and not whether a flow actually features an HTTP request, we use the TCP `PSH` flag. This flag signals an application-layer data exchange, so we consider a new connection to a common Web server port (80, 443) to feature an HTTP request. We filter out every flow that does not feature at least TCP `SYN`, `ACK`, `PSH` and `FIN`/`RST` flags. The retrieved data is presented to the next step, *clustering*, per tuple.

4.3 Clustering

Histograms of attack traffic are very much alike when attacks reside in the *brute-force* phase, since repeated application-layer behavior results in flat traffic, as shown in Sect. 3. As a consequence, this will cause *brute-force* traffic to be clustered. For the *compromise* phase, however, we analyze precisely the traffic that falls outside the cluster featuring *brute-force* phase histograms. We avoid any clustering impairment caused by TCP protocol variability (e.g., control information segments), as described in [14], by removing the zero-value bins from histograms; potential traffic variability is typically caught in zero-value bins as related segments do not feature any payload.

In this work, we use Hierarchical Cluster Analysis (HCA), which aims at building clusters based on inter-cluster distances in a hierarchical fashion [15]. HCA uses either one of the following strategies: *divisive* (also commonly referred to as *top-down*), or *agglomerative* (*bottom-up*). We take the agglomerative approach, because it is faster than divisive clustering for larger datasets if the entire hierarchy needs to be built. Since histograms describing compromises are assumed to be outliers compared to histograms describing brute-force traffic, we would need to 'divide' all the way down to individual histograms to find potential 'compromise outliers' in case of divisive clustering.

The advantage of using HCA is that, unlike other clustering approaches such as *k*-means, HCA does not require the number of clusters to be set in advance. Instead, HCA can stop the clustering process (referred to as *stop linking clusters* in HCA jargon) as soon as certain criteria are no longer satisfied. An example criterion is that the distance between the selected pair of observations for cluster linkage is above a given threshold. Linking clusters is done based on a linkage method and a distance metric. The linkage method determines which two histograms from two potentially to be linked clusters to apply the distance metric to (i.e., which histograms to use for inter-cluster distance calculation), as each of the two clusters potentially contains multiple histograms. In this work, we rely on *single-linkage* as the linkage method, and on MDPA as the distance metric, as discussed in Sect. 3. Single-linkage selects the two least dissimilar histograms in two clusters to

determine the inter-cluster distance. The choice for single-linkage, as opposed to *complete-linkage* (which considers the two most dissimilar histograms), was made empirically based on clustering results for datasets that were confirmed to feature Web attacks.

To find the optimum number of clusters, we express the validity of clustering results in terms of an 'internal index' after every HCA step. This index indicates how well observations, i.e., histograms, lie within their cluster, and how well clusters are distanced. As opposed to an external index, an internal index does not require external information (e.g., ground truth) for validation, which is desirable in our case because we need to account for diverse datasets. As shown in [22], a cluster can be graphically represented by its so-called *Silhouette*, which is composed of the Silhouette coefficients of each observation in that cluster. Silhouette coefficients are mathematically defined as follows:

$$s(x) = \frac{b(x) - a(x)}{max(a(x), b(x))} \tag{5}$$

In our context, $a(x)$ represents the average dissimilarity of histogram X to all other objects within the same cluster, while $b(x)$ is the inter-distance of histogram X to its neighboring cluster (i.e., the second-best cluster choice for histogram X). Silhouette coefficients are particularly interesting for our approach, since they can be calculated based only on pair-wise distances between observations in the dataset, for which we can use the MDPA metric. The average of all Silhouette coefficients lends itself well as an internal index, and as such can be used to validate clustering results and determine the optimum number of clusters [22]. To this end we have also studied alternative internal indices, such as the one used in Calinski and Harabasz (CH) [3], which is based on a 'sum-of-squares'. Unlike the average Silhouette coefficient, the CH index is based on the distance of the cluster centroids to the general mean of the data [16]. Since the MDPA metric does not allow for the mean between more than two histograms to be determined, neither cluster centroids, nor the general mean can be calculated. This yields not only the CH approach infeasible in combination with MDPA, but any other internal index that relies on 'sums-of-squares'.

4.4 *Brute-Force* Phase Detection

Detection of the *brute-force* phase is always done using the largest cluster of a tuple, since we assume that attack traffic is dominant enough to comprise a (large) cluster by itself. In cases of non-attack traffic, the largest cluster may however contain histograms that are not very similar, meaning that distances between histograms are rather large. To filter out such candidates, we calculate the average intra-cluster distance for the largest cluster. In case it exceeds a predefined threshold θ,[5] we ignore the tuple in the remainder of the detection procedure.

[5] We use $\theta = 3$, meaning that a cluster's histograms must be roughly identical, i.e., three bytes deviation on average. This value was empirically established based on the analysis of real attacks.

Several CMSs, such as recent versions of Joomla and Drupal, have built-in mechanisms to mitigate simple brute-force attacks against their backends, mostly by requiring a token, served by the CMS as part of a session cookie or a form nonce, to be included in authentication requests. Depending on the attacked CMS, the token(s) required for authentication may have to be retrieved only once per attack (Joomla) or once per authentication request (Drupal). The thought behind this is that not-so-clever brute-force tools start their attacks without retrieving the authentication pages first, causing the attacks to never yield any useful result. However, our analysis of modern attack tools has revealed that they are perfectly able to circumvent this type of protection nowadays, which will be reflected in the overall cluster structure. Therefore, besides analyzing merely the largest cluster, we analyze the relation between the two largest clusters and verify whether they have the following characteristics:

- Clusters must feature a typical relation in terms of size, such as 1:1, 1:2 or 1:3.
- The average distance between histograms in both clusters, γ, must be at least 75. This value has been established empirically based on our datasets and our analysis has shown that many CMSs will trigger such behavior with a distance of only around 200. This distance can be explained by the completely different nature of requests for token retrieval and authentication attempts. Also, this minimum is used to avoid considering two clusters that have somewhat similar histograms which should not have been separated into these two clusters to begin with.

Given the alternating nature of token retrieval (typically done using HTTP GET requests) and authentication attempts (using HTTP POST), we refer to this behavior as *GET/POST-alternation* or *GET/GET/POST-sequence*.

Finally, once the largest cluster is found to feature a *brute-force* phase, we perform a sanity check to rule out false positives: in case the set of clusters features many small clusters, i.e., clusters with only one or two histograms, we 'overrule' the detection of the *brute-force* phase. Many small clusters indicate that the network traffic was highly variable in terms of payload, therefore contradicting our definition of typical attack behavior.

4.5 Compromise Phase Detection

Authentication attempts that ultimately result in a compromise are no different from connections resulting in failed authentication attempts, as the request sent to the Web server is basically identical (except for the credentials themselves). We therefore analyze return traffic, i.e., traffic from target to attacker, to identify potential compromises. Our analysis of attacks has shown that these return connections are different in size upon successful authentication. The difference in size can be explained by the (new) page, e.g., CMS backend panel, that is served to the attacker, which is different from the login forms used in the *brute-force* phase.

Since compromises can only be present after login attempts, an attack must reside in the *brute-force* phase before the *compromise* phase detection is activated. To

detect compromises, we retrieve flow data from target to attacker and cluster the payload histograms in exactly the same way as for the *brute-force* phase detection. Since traffic from target to attacker consists of many authentication errors in a typical brute-force attack, we assume that traffic to be rather alike in terms of its payload distribution. If there exists only one cluster with a single histogram (referred to as the *single-histogram cluster* in the remainder of this section), we continue the detection. Otherwise, many small clusters point to traffic that is scattered in terms of payload, while clusters with more histograms are unlikely to feature a compromise, because the compromise should be considered an 'exceptional' case within the attack.

Based on measurements in a lab environment, where we have recorded the network traffic between the attack tool *Patator* and the three CMSs considered in this work, we have identified various characteristics of a compromise. As such, if a histogram in the single-histogram cluster matches the following two criteria, we consider it a compromise:

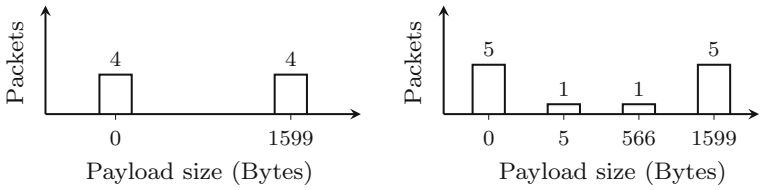
- The histogram 'size', i.e., each bin multiplied by its value, is larger than the average size of all other clusters.
- The histogram's bins must differ from bins in all other clusters, since we assume that the flow for the compromise carries significantly different traffic, such as a CMS backend management page. The only exceptions are the zero-value bins, which are ignored from our detection, as explained in Sect. 4.3, and the bins that represent the maximum segment size (MSS), as they are also likely present in both *brute-force* phase and *compromise* phase histograms.

These characteristics are visualized in Fig. 8, where we show both a *brute-force* phase histogram and a *compromise* phase histogram based on our measurements. It is clear that the overall size of the *compromise* phase histogram is larger than the size of the *brute-force* phase histogram and that the histogram's bins are different. It should be noted that the *brute-force* phase obviously consists of a whole bunch of histograms like the one shown in Fig. 8a, rather than only one.

5 Validation

In the context of our validation, we define an *attack* as a sequence of at least N flows towards a CMS backend. Due to the fact that connections that are part of an attack feature at least one authentication attempt, N consecutive flows feature at least N authentication attempts. By measuring the number of consecutive connections towards the same CMS backend Uniform Resource Locator (URL), we have found that $N = 20$ covers roughly the upper 10% of attack sizes measured in our validation datasets. This value, on the one hand, causes benign failed login attempts to be filtered out implicitly, and, on the other hand, reduces the load on our prototype due to very small attacks and noise.

The remainder of this section is structured as follows. In Sect. 5.1, we introduce the two prototypes that are developed for demonstrating the contributions of this



(a) *Brute-force* phase. **(b)** *Compromise* phase.

Fig. 8 Histograms for flows in various phases of an attack against a vanilla Drupal instance. **a** *Brute-force* phase. **b** *Compromise* phase

work. Then, in Sect. 5.2, we explain the datasets that have been collected and analyzed for validation. This is followed by an introduction of our validation approach and the applied metrics in Sects. 5.3 and 5.4, respectively, followed by a discussion of the validation results in Sect. 5.5.

5.1 Prototypes

For the sake of validating our detection approach, we have implemented two prototypes. First, we have modified the IPFIX Metering and Exporting processes of our flow exporter, such that it is able to export payload size histograms for every observed/metered flow using IPFIX. INVEA-TECH's FlowMon platform was chosen for this purpose, as it has been designed with extensibility in mind. Nevertheless, our extension can easily be ported to other flow exporters, such as nProbe[6] and YAF.[7] Second, we have implemented an intrusion detection prototype that performs the preselection and detection as described in Sect. 4.

5.2 Datasets

The datasets used for validation of this work have been collected in the production network of Hosting 2GO, a Top-10 Web-hosting company in the Netherlands, for a period of a month in July/August 2015. The systems under observation host approximately 2500 Web hosting accounts, worth a total of 2603 *vhosts*. In total, the datasets consist of traffic records worth 414 GB, generated by more than 237*k* hosts. More specifically, we use two types of datasets, summarized in Table 1, both from the same observation point, but collected on different systems:

- *Log files*—These Web server access log files serve as ground-truth for our validation and are the only means of verifying whether an attacker has compromised a system.
- *Flow data*—This data has been exported using IPFIX with 1:1 sampling applied on the exporting device, and consists of the typical set of fields seen in many NetFlow v9 implementations. Among those fields are IP addresses and port numbers, L3 protocol number, IP Type of Service (ToS) byte and Simple

[6] http://www.ntop.org/products/netflow/nprobe/

[7] https://tools.netsa.cert.org/yaf/

Network Management Protocol (SNMP) input interface ID. Additionally, we augment the exported data with HTTP information, namely hostname and URL, and payload metadata, as discussed in Sect. 3. The functionality for parsing hostnames in HTTP and HTTPS traffic, and URLs in HTTP traffic is available on every modern FlowMon device. The code for exporting payload size histograms is our own. Our flow data serves as input for the prototype described in Sect. 5.1.

To protect the privacy of individuals of whom we have captured network traffic and log files, we have anonymized all fields that can potentially lead to personal identification. First and foremost, we have anonymized all IP addresses in both the flow data and log files in a *prefix-preserving* manner using the de facto standard in this area: Crypto-PAn. Crypto-PAn is consistent across traces, such that we can correlate IP addresses in both datasets, even after anonymization. Second, we have hashed all HTTP hostnames (i.e., *vhosts*), such that the original hostname cannot be retrieved anymore, while hostnames can still be uniquely identified in both datasets.

5.3 Approach

Out of the three attack phases discussed in Sect. 2, only the *scan* phase has not been touched in this work, due to its irrelevance in the context of attacks against Web applications. We therefore validate our detection performance only with respect to the *brute-force* and *compromise* phases. Additionally, to underline the improvements of our approach compared to the state-of-the-art, we also compare our results to results obtained based on the approach described in [24].

Before we can compare detection results to our ground-truth, we have to post-process the datasets to obtain the same unit of comparison, as shown in Fig. 9. For this purpose, we define an *interaction* as a set of consecutive sessions/connections towards a CMS backend within a certain time period. To obtain interactions from Web server log files, we developed an Apache access log parser that aggregates log records into interactions based on a number of heuristics. In the case of flow data, which naturally consists of session/connection entries, the only post-processing needed is the aggregation of sessions between a tuple with less than 5 min of idle time between sessions. This idle time was chosen as a tradeoff between the typical, very short-lived HTTP(S) sessions between server and client on the one hand, and a buffer for coping with time offsets between datasets on the other. After post-processing, our ground-truth consists of 854945 interactions.

As for identifying brute-force attacks and compromises from log files, we take the following approaches. For the *brute-force* phase, we determine the number of consecutive HTTP POST requests towards CMS backend login pages. If this number exceeds N, we classify the interaction as malicious. For the *compromise* phase, the general approach is to label an interaction as to feature a compromise as soon as an attacker is not interacting with the login form anymore, which however strongly varies per CMSs. WordPress is the most simple case, since it uses different URLs for login page and backend. Joomla's backend page after login is greater in size than the login page itself, so one or few larger responses by the Web server

Table 1 Validation datasets

Dataset	Size on disk (GB)	Size (entries)
Access logs	2.8	12.2 M
Flow data	16.8	42.1 M

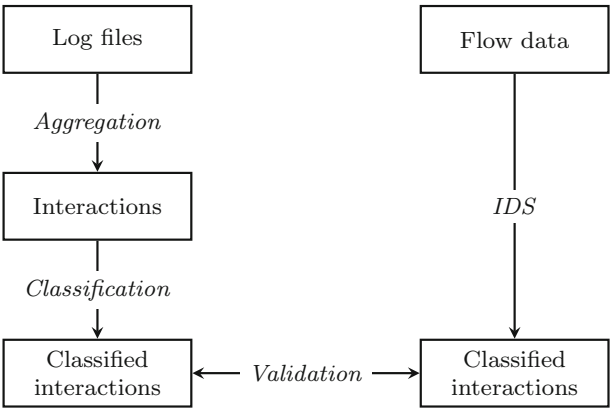


Fig. 9 Dataset post-processing for validation

signal a login. In the case of Drupal, a successful login is indicated by a different HTTP status code, which performs a redirection and thus carries less HTTP payload.

5.4 Metrics

IDS performance metrics are typically expressed in terms of positive and negative detections being either true or false. We therefore define metrics for the *brute-force* phase as follows:

- True Positive $(TP)_B$: Interaction labeled as malicious that is reported by our prototype.
- False Positive $(FP)_B$: Interaction labeled as benign that is reported by our prototype.
- True Negative $(TN)_B$: Interaction labeled as benign that is not reported by our prototype.
- False Negative $(FN)_B$: Interaction labeled as malicious that is not reported by our prototype.

These metrics can also be expressed as percentages. For example, the True Positive Rate (TPR) is defined in the context of this work as the percentage of interactions correctly labeled as malicious and reported by our prototype:

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

To avoid any bias of *brute-force* phase detection results on detection results for the *compromise* phase, we only consider those attacks that were successfully found to feature a *brute-force* phase. This is a logical consequence of the fact that the *compromise* phase can only be reached after the *brute-force* phase, as described in Sect. 2. From the ground truth, we conclude a compromise upon change of a URL between a tuple from a backend authentication URL to another URL on the same *vhost*, after more than N authentication attempts. As for our prototype, we measure its performance in terms of the following evaluation metrics for the *compromise* phase:

- TP_C: TP_B correctly identified to feature a compromise.
- FP_C: TP_B incorrectly identified to feature a compromise.
- TN_C: TP_B correctly identified to not feature a compromise.
- FN_C: TP_B incorrectly identified to not feature a compromise.

Additionally, we use the aforementioned evaluation metrics to calculate the accuracy (Acc) of our prototype:

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$

We have found some (positive and negative) detection results for the *brute-force* phase that could not be matched with our ground-truth, mostly because of timing deviations between our datasets. To make sure that these detections are not accounted wrongly in any of our evaluation metrics, we have listed them separately as *unclassified*.

5.5 Results

Our validation results are shown in Tables 2 and 3, which lists the values for all evaluation metrics, as well as their respective rates/percentages. The most important take-away is that our approach (Table 2) significantly outperforms existing works (Table 3), in all respects. Besides detecting a significantly larger number of TP_B (469 vs. 237), the number of false detections has been reduced to almost 10% of existing works. Also, our approach has shown to be able to detect the one compromise in the dataset. It should be noted that we validated our work in a conservative case, because our validation network has a firewall in place that blocks remote hosts when generating too many connections. Consequently, (very) large attacks have never reached the Web servers and are therefore not recorded in our datasets, while they would likely be detected by our prototype.

With respect to false detections, we made several observations. First, almost all FPs are somehow related to photo galleries. Photos in a gallery are often similar in size, because they have been shot by the same camera and post-processed in the same way. They may even have been compressed such that they end up having the

Table 2 Validation results for our approach

Phase	TP	FP	TN	FN	*Acc*	*Uncl.*
Brute-force	469 (0.597)	519 (0.000)	1.748M (1.000)	317 (0.403)	1.000	21,336
Compromise	1 (1.000)	14 (0.030)	454 (0.970)	0 (0.000)	0.970	0

Table 3 Validation results for existing works

Phase	TP	FP	TN	FN	*Acc*	*Uncl.*
Brute-force	237 (0.303)	5058 (0.003)	1.744M (0.997)	545 (0.697)	0.997	21382
Compromise	–	–	–	–	–	–

same size. Thumbnails are even worse (for our approach); Once an album is opened, thumbnails of the album's contents are fetched by the client, resulting in tons of similar connections, due to the fact that the thumbnails have exactly the same dimensions and are typically identical in size. Second, FNs have two major causes: either they are caused by low-intensity attacks that do not generate more than N connections/requests per 5 min data chunk, or the distance between histograms in the largest cluster slightly exceeds our threshold θ.

6 Discussion

This work so far presented a detection approach for identifying brute-force attacks in flow data and potential compromises resulting thereof, without referencing any mitigation possibilities. A straightforward mitigation approach is blacklisting; by adding identified attackers to access control lists (ACLs) in a firewall or router, any traffic from these sources is blocked. However, although our prototype has shown to report just a small number of false detections, we propose a less radical mitigation approach: graylisting. As with typical blacklisting, graylisting works by listing IP address of attackers in a router or firewall, but instead of pure blocking, attackers are redirected to a static landing page. This page allows one to be delisted by clicking a button or entering a keyword, and since it is not recognized by attack tools, humans can easily be unblocked, while automated attacks are mitigated.

Several Top-10 Web hosting companies in the Netherlands mitigate attacks by serving a CAPTCHA or HTTP Basic Authentication ahead of every CMS backend login page, enforced on a per-server basis. Our analysis of attack tools has shown that tools do not recognize this behavior and naively continue an attack. We want to point out that our approach still works in such environments, since attacks will feature many connections that are similar in terms of their payload distribution.

7 Conclusions

A lesson learned from 25 years of network and systems management research, is that the focus has changed from designing new management technologies towards solving real management problems. One key problem managers are faced with today, is the detection of security attacks on their infrastructure. In this paper we discuss how to detect brute-force attacks and subsequent compromises of Web applications. Traditionally such detection is performed on the Web server hosts (systems management) by analysing log files. However, in environments where hosting companies do not have complete access to all log files, for example in cases where customers are offered VMs, such approach may no longer be possible. This paper therefore discusses the feasibility of a network-based approach, in which IPFIX data is analysed to detect compromises.

The use of clustering methods together with histograms for discriminating between attack traffic and non-attack traffic makes our approach applicable to almost any Web application (Sect. 4). In addition, it allows us to overcome the problems faced by related works, e.g., false negatives due to TCP control information, and false positives as a consequence of benign applications generating many similar connections, such as Web crawlers (Sect. 3).

Although the use of histograms increases the size per flow record and therefore the amount of data to be transferred and processed, we show that their impact is very limited; our histograms extend the size of flow records by at most 37 bytes in 99% of observed cases (Sect. 3). This number can certainly be optimized if circumstances dictate, by using binary encoding, for example. Our validation results show that, although the number of compromises in our datasets is very limited, we are able to detect all of them (Sect. 5). In addition, we significantly outperform related works by roughly halving the number of false detections.

There are still several directions for future work. First and foremost, like most security measurement studies, false positives and negatives remain an important issue. For this specific study more work is needed to understand the most dominant sources of false detections, such as online galleries with thumbnails. Since retrieving thumbnails is typically done in batches, e.g., when opening a Web page, we expect that timing characteristics may be useful for discriminating traffic towards galleries from attack traffic. Second, different cipher suites and key exchange protocols will affect the size of HTTPS flows. Attackers could try to use different cipher suites and protocols in an attempt to remain invisible. In general it is important to make detection approaches more resilient against evasion techniques.

Appendix 1: Minimum Difference of Pair Assignment (MDPA)

MDPA is a distance metric for calculating how similar two histograms are. Its most distinguishing feature is that it does not satisfy the *shuffling invariance* metric, meaning that shuffling any bin values in a histogram *does* affect histogram inter-distance, as explained by means of an example in Sect. 3. In short, MDPA aims at finding the minimum difference of pair assignments between two sets, histograms in our context. As such, one has to find the best combination of one-to-one assignments such that the sum of all differences is as small as possible [4].

Calculation

The formal definition of the inter-distance of two histograms using the MDPA metric has been defined in (4). How the actual calculation works can best be explained based on Fig. 10, where Histogram A, B and C are taken from Fig. 3. Distances between any two histograms are the sums of differences between pairs of samples. For example, the difference between Histogram A and B is 4. The minimum distance of pairs between Histogram A and C is 204. To illustrate the effect of MDPA not satisfying the *shuffling invariance*, we have shuffled the samples in Histogram C in Fig. 10 such that the samples are not in ascending order of their values anymore. Given that all distance permutations are compared in (4), the overall distance between any two histograms is not affected by the order of the samples.

Normalization

Since the number of samples in a histogram is not necessarily the same between any two histograms, one can apply normalization of the histograms by multiplying all elements by a common multiple N of both histograms. One common multiple is the product of the number of samples in two histograms x and y, i.e., $n_x * n_y$. The normalized distance between two histograms is then defined as follows [4]:

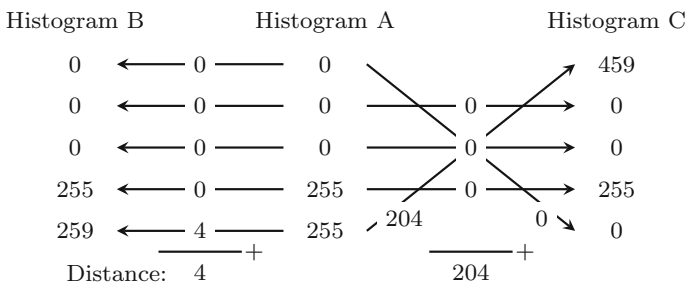


Fig. 10 MDPA calculation. Visualization based on [4]

$$D^N(x,y) = \frac{D(x^N, y^N)}{N} \tag{8}$$

Although normalization is only necessary in case of inequality of the number of samples in any set of histograms, we use the normalized distance in all our calculations for the following reasons:

- It is very unlikely that histograms within a cluster are identical in terms of the number of featured samples.
- Comparing histogram distances between different clusters can only be done if the distances are normalized.

References

1. Best Host News: cPanel vs. Plesk vs. DirectAdmin comparison. https://www.besthostnews.com/cpanel-vs-plesk-vs-directadmin/ (2015). Accessed 9 June 2017
2. Burnett, M.: Yes, 123456 is the most common password, but here's why that's misleading. http://arstechnica.com/security/2015/01/yes-123456-is-the-most-common-password-but-heres-why-thats-misleading/ (2015). Accessed 9 June 2017
3. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Commun. Stat. Theory Methods **3**(1), 1–27 (1974)
4. Cha, S.H., Srihari, S.N.: On measuring the distance between histograms. Pattern Recognit. **35**(6), 1355–1370 (2002)
5. Cid, D.: WordPress malware—active VisitorTracker campaign. https://blog.sucuri.net/2015/09/wordpress-malware-active-visitortracker-campaign.html (2015). Accessed 9 June 2017
6. Claise, B., Trammell, B., Aitken, P.: Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information. RFC 7011 (Internet Standard). http://www.ietf.org/rfc/rfc7011.txt (2013)
7. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: an empirical analysis. Proc. IEEE INFOCOM **2010**, 1–9 (2010)
8. Drašar, M.: Protocol-independent detection of dictionary attacks. In: Proceedings of the 19th EUNICE/IFIP WG 6.6 International Workshop, EUNICE'13, pp. 304–309 (2013)
9. Drašar, M.: Behavioral detection of distributed dictionary attacks. Ph.D. thesis, Masaryk University, Brno, Czech Republic (2015)
10. Gooding, S.: 100,000+ WordPress sites compromised using the slider revolution security vulnerability. http://wptavern.com/100000-wordpress-sites-compromised-using-the-slider-revolution-security-vulnerability (2014). Accessed 9 June 2017
11. Hofstede, R., Hendriks, L., Sperotto, A., Pras, A.: SSH compromise detection using NetFlow/IPFIX. ACM SIGCOMM Comput. Commun. Rev. **44**(5), 20–26 (2014)
12. Huckaby, J.: How to scan WordPress like a hacker. http://www.rackaid.com/blog/scan-wordpress/ (2014). Accessed 9 June 2017
13. Javed, M., Paxson, V.: Detecting stealthy, distributed SSH brute-forcing. In: Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security, CCS'13, pp. 85–96 (2013)
14. Jonker, M., Hofstede, R., Sperotto, A., Pras, A.: Unveiling flat traffic on the internet: an SSH attack case study. In: Proceedings of the 14th IFIP/IEEE Symposium on Integrated Network and Service Management, IM'15 (2015)
15. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis, vol. 344. Wiley, Hoboken (2009)
16. Kind, A., Stoecklin, M.P., Dimitropoulos, X.: Histogram-based traffic anomaly detection. IEEE Trans. Netw. Serv. Manag. **6**(2), 110–121 (2009)

17. Koch, R.H.: Systemarchitektur zur Ein- und Ausbruchserkennung in verschlüsselten Umgebungen. Ph.D. thesis, Universität der Bundeswehr München, München, Germany (2015)
18. Mekky, H., Torres, R., Zhang, Z.L., Sabyasachi, Nucci, A.: Detecting malicious HTTP redirections using trees of user browsing activity. In: Proceedings of IEEE INFOCOM 2014, pp. 1159–1167 (2014)
19. Perez, T.: Understanding denial of service and brute force attacks—WordPress, Joomla, Drupal, vBulletin. https://blog.sucuri.net/2014/03/understanding-denial-of-service-and-brute-force-attacks-wordpress-joomla-drupal-vbulletin.html (2014). Accessed 9 June 2017
20. Piskac, P., Novotny, J.: Using of time characteristics in data flow for traffic classification. In: Proceedings of the 5th International Conference on Autonomous Infrastructure, Management and Security, AIMS 2011. Lecture Notes in Computer Science, vol. 6734, pp. 173–176. Springer, Berlin (2011)
21. Qiu, H., Eklund, N., Hu, X., Yan, W., Iyer, N.: Anomaly detection using data clustering and neural networks. In: Proceedings of the IEEE International Joint Conference on Neural Networks, 2008, IJCNN'08, pp. 3627–3633 (2008)
22. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**, 53–65 (1987)
23. Sucuri: WordPress brute force attacks. https://sucuri.net/security-reports/brute-force/ (2015). Accessed 9 June 2017
24. van der Toorn, O., Hofstede, R., Jonker, M., Sperotto, A.: A first look at HTTP(S) intrusion detection using NetFlow/IPFIX. In: Proceedings of the 14th IFIP/IEEE Symposium on Integrated Network and Service Management, IM'15, pp. 862–865 (2015)
25. Vizváry, M., Vykopal, J.: Flow-based detection of RDP brute-force attacks. In: Proceedings of 7th International Conference on Security and Protection of Information, SPI'13, pp. 131–137 (2013)
26. Vykopal, J.: Flow-based brute-force attack detection in large and high-speed networks. Ph.D. thesis, Masaryk University, Brno, Czech Republic (2013)
27. Vykopal, J., Plesnik, T., Minarik, P.: Network-based dictionary attack detection. In: Proceedings of 2009 International Conference on Future Networks, ICFN'09, pp. 23–27 (2009)
28. Walker, D.: Botnet of Joomla servers furthers DDoS-for-hire scheme. http://www.scmagazine.com/ddos-campaign-exploits-servers-with-vulnerable-google-maps-plug-in/article/400473/ (2015). Accessed 9 June 2017

Rick Hofstede is a former Ph.D. student of the University of Twente and the University der Bundeswehr München, Germany. In 2016, he successfully defended his Ph.D. thesis titled "Flow-based Compromise Detection", after which he switched to the cyber security industry. His main areas of interest include cyber security, big data processing and network forensics.

Mattijs Jonker received the B.Sc. and M.Sc. degrees in Computer Science from the University of Twente, The Netherlands, where he is currently pursuing the Ph.D. degree within the Design and Analysis of Communication Systems group. The focus of his research is DDoS attack mitigation. His main research interests include network security, Internet measurements and big data analytics.

Anna Sperotto is assistant professor at the Design and Analysis of Communication Systems Group of the University of Twente, the Netherlands. She received a Ph.D. degree from the University of Twente, in 2010, with the thesis titled "Flow-based intrusion detection". Her research interests include network security, network measurements and traffic monitoring and modeling.

Aiko Pras is a professor in the area of Network Operations and Management at the University of Twente, The Netherlands. His research interests include network management technologies, network monitoring, measurements and security. He was chair of IFIP TC6 and coordinator of the European FP7 FLAMINGO project.