PUBCRAWL: Protecting Users and Businesses from CRAWLers

Gregoire Jacob
University of California, Santa Barbara / Telecom SudParis
gregoire.jacob@gmail.com

Engin Kirda
Northeastern University
ek@ccs.neu.edu

Christopher Kruegel
University of California, Santa Barbara
chris@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
vigna @cs.ucsb.edu

Abstract

Web crawlers are automated tools that browse the web to retrieve and analyze information. Although crawlers are useful tools that help users to find content on the web, they may also be malicious. Unfortunately, unauthorized (malicious) crawlers are increasingly becoming a threat for service providers because they typically collect information that attackers can abuse for spamming, phishing, or targeted attacks. In particular, social networking sites are frequent targets of malicious crawling, and there were recent cases of scraped data sold on the black market and used for blackmailing.

In this paper, we introduce PUBCRAWL, a novel approach for the detection and containment of crawlers. Our detection is based on the observation that crawler traffic significantly differs from user traffic, even when many users are hidden behind a single proxy. Moreover, we present the first technique for crawler campaign attribution that discovers synchronized traffic coming from multiple hosts. Finally, we introduce a containment strategy that leverages our detection results to efficiently block crawlers while minimizing the impact on legitimate users. Our experimental results in a large, well-known social networking site (receiving tens of millions of requests per day) demonstrate that PUBCRAWL can distinguish between crawlers and users with high accuracy. We have completed our technology transfer, and the social networking site is currently running PUBCRAWL in production.

1 Introduction

Web crawlers, also called spiders or robots, are tools that browse the web in an automated and systematic fashion. Their purpose is to retrieve and analyze information that is published on the web. Crawlers were originally developed by search engines to index web pages, but have since multiplied and diversified. Crawlers are now used as link checkers for web site verification, as scrapers to harvest content, or as site analyzers that process the collected data for analytical or archival purposes [9]. While many crawlers are legitimate and help users find relevant content on the web, unfortunately, there are also crawlers that are deployed for malicious purposes.

As an example, cyber-criminals perform large-scale crawls on social networks to collect user profile information that can later be sold to online marketers or used for targeted attacks. In 2009, a hacker who had crawled the popular student network StudiVZ, blackmailed the company, threatening to sell the stolen data to gangs in Eastern Europe [23]. In 2010, Facebook sued an entrepreneur who crawled more than 200 million profiles, and who was planning to create a third-party search service with the data that he had collected [25]. In general, the problem of site scraping is not limited to social networks. Many sites who advertise goods, services, and prices online desire protection against competitors that use crawlers to spy on their inventory. In several court cases, airlines (e.g., American Airlines [2], Ryanair [20]) sued companies that scraped the airlines' sites to be able to offer price comparisons and flights to their customers. In other cases, attackers scraped content from victim sites, and then simply offered the cloned information under their own label.

Many websites explicitly forbid unauthorized scraping in their terms of services. Unfortunately, such terms are simply ignored by non-cooperating (malicious) crawlers. The *Robot Exclusion Protocol* faces a similar problem: Web sites can specify rules in the `robots.txt` file to limit crawler accesses to certain parts of their site [14], but a crawler has to voluntarily follow these restrictions.

Current detection techniques rely on simple crawler artifacts (e.g., spurious user agent strings, suspicious referrers) and simple traffic patterns (e.g., inter-arrival time, volume of traffic) to distinguish between human and crawler traffic. Unfortunately, better crawler implementations can remove revealing artifacts, and simple traffic patterns fail in the presence of proxy servers or large corporate gateways, which can serve hundreds of legitimate users from a single IP address. In response to the perceived lack of effective protection, several commercial anti-scraping services have emerged (e.g., *Dis-*

til.It, *SiteBlackBox*, *Sentor Assassin*). These services employ "patent pending heuristics" to defend against unwanted crawlers. Unfortunately, it is not clear from available descriptions how these services work in detail.

Many sites rely on CAPTCHAs [24] to prevent scrapers from accessing web content. CAPTCHAs use challenge-response tests that are easy to solve for humans but hard for computers. Some tests are known to be vulnerable to automated breaking techniques [4]. Nevertheless, well-designed CAPTCHAs offer a reasonable level of protection against automated attacks. Unfortunately, their excessive use brings along usability problems and severely decreases user satisfaction. Other prevention techniques are crawler traps. A crawler trap is a URL that lures crawlers into infinite loops, using, for example, symbolic links or sets of auto-generated pages [15]. Unfortunately, legitimate crawlers or users may also be misled by these traps. Traps and CAPTCHAs can only be one part of a successful defense strategy, and legitimate users and crawlers must be exposed as little as possible to them.

In this paper, we introduce a novel approach and a system called PUBCRAWL to detect crawlers and automatically configure a defense strategy. PUBCRAWL's usefulness has been confirmed by a well-known, large social networking site we have been collaborating with, and it is now being used in production. Our detection does not rely on easy-to-detect artifacts or the lack of fidelity to web standards in crawler implementations. Instead, we leverage the key observation that crawlers are automated processes, and as such, their access patterns (web requests) result in different types of regularities and variations compared to those of real users. These regularities and variations form the basis for our detection.

For detection, we use both content-based and timing-based features to *passively* model the traffic from different sources. We extract content-based features from HTTP headers (e.g., referrers, cookies) and URLs (e.g., page revisits, access errors). These features are checked by heuristics to detect values betraying a crawling activity. For timing-based features, we analyze the time series produced by the stream of requests. We then use machine learning to train classifiers that can distinguish between crawler and user traffic. Our system is also able to identify crawling campaigns led by distributed crawlers by looking at the synchronization of their traffic.

The aforementioned features work well for detecting crawlers that produce a minimum volume of traffic. However, it is conceivable that some adversaries have access to a large botnet with hundreds of thousands of infected machines. In this case, each individual bot would only need to make a small number of requests to scrape the entire site, possibly staying below the minimal volume required by our models. An *active* response to such attacks must be triggered, such as the injection of crawler traps or CAPTCHAs. An active response is only triggered when a single client sends more than a (small) number of requests. To minimize the impact of active responses on legitimate users, we leverage our detection results to distinguish between malicious crawlers and benign sources that produce a lot of traffic (e.g., proxy servers or corporate gateways). This allows us to automatically whitelist benign sources (whose IP addresses rarely change), minimizing the impact on legitimate users while denying access to unwanted crawlers.

For evaluation, we applied PUBCRAWL to the web server logs of the social networking site we were working with. To train the system, we examined 5 days worth of traffic, comprising 73 million requests from 813 average-volume sources (IP addresses). The logs were filtered to focus on sources whose traffic volume was not an obvious indicator of their type. To test the system, we examined a set of 62 million requests coming from 763 sources over 5 days. Our experiments demonstrated that more sophisticated crawlers are often hard to distinguish from real users, and hence, are difficult to detect using traditional techniques. Using our system, we were able to identify crawlers with high accuracy, including crawlers that were previously-unknown to the social networking site. We also identified interesting campaigns of distributed crawlers.

Section 2 gives an overview of the system whereas Sections 3 to 5 provide more technical details for each part. The configuration and evaluation of the system if finally addressed in Sections 6 to 8. Overall, this paper makes the following contributions:

- We present a novel technique to detect individual crawlers by time series analysis. To this end, we use auto-correlation and decomposition techniques to extract navigation patterns from the traffic of individual sources.
- We introduce the first technique to detect distributed crawlers (crawling campaigns). More precisely, our system can identify coordinated activity from multiple crawlers.
- We contain crawlers using active responses that we strategically emit according to detection results.
- We implemented our approach in a tool called PUBCRAWL, and performed the largest real-world crawler detection evaluation to date, using tens of millions of requests from a popular social network. PUBCRAWL distinguishes crawlers from users with high accuracy (even users behind a proxy).

2 System Overview

The goal of PUBCRAWL is to analyze the web traffic that is sent to a destination site, and to automatically classify

the originating sources of this traffic as either crawlers or users. The initial traffic analysis is passive and performed off-line, using log data that records web site requests from clients. The goal of this analysis is to build a knowledge base about traffic sources (IP addresses).

This knowledge base is then consulted to respond to web requests. Requests from known users or accepted crawlers (e.g., *Googlebot*) are served directly. Requests from unwanted crawlers, in contrast, are blocked. Of course, the knowledge base may not contain an entry for a source IP. In this case, a small number of requests is permitted until it exceeds a given threshold: the system then switches to active containment and, for example, injects traps or CAPTCHAs into the response.

While the system is active, the requests are recorded to refine the knowledge base: When PUBCRAWL identifies a previously-unknown source to be a user or a legitimate proxy, requests from this source are no longer subjected to active responses whereas unwanted crawlers are blacklisted. The key insight is that PUBCRAWL can successfully identify legitimate, high-volume sources, and these sources are very stable. This stability of high-volume sources and the large number of low-volume sources (tens of requests) ensure that only a small fraction of users will be subjected to active responses.

The architecture of PUBCRAWL is shown in Figure 1. The server log entries (i.e., the input) are first split into time windows of fixed length. Running over these windows, the system extracts two kinds of information: *(i)* HTTP header information, including URLs and *(ii)* timing information in the form of time series. The extracted information is then submitted to two detection modules (which detect individual crawlers) and an attribution module (which detects crawling campaigns). These three modules generate the knowledge base that is leveraged by the proactive containment module for real-time traffic. The different modules are described below:

Heuristic detection. For a given time window, the system analyzes the requests coming from distinct source IP addresses, and extracts different features related to HTTP header fields and URL elements. These features are checked with heuristics to detect suspicious values that could reveal an ongoing crawling activity (e.g., suspicious referrers, unhandled cookies, etc.).

Traffic shape detection. When crawlers correctly set the different HTTP fields and masquerade the user agent string, it becomes much more difficult to tell apart slow crawlers from busy proxies since they cannot be distinguished based on request volumes. Traffic shape detection addresses this problem. For a given source IP address, the system analyzes the requests (time stamps) over a fixed time window to build the associated time series. Figure 2 depicts time series representing crawler

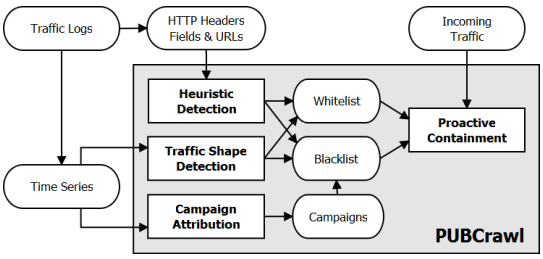


Figure 1: Architecture overview

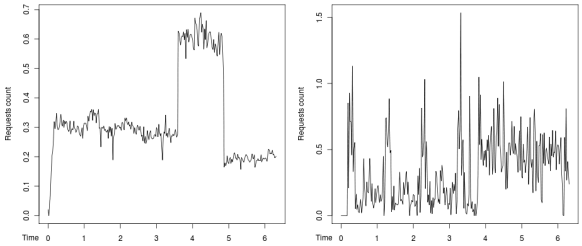


Figure 2: YahooSlurp and MSIE 7 time series

and user traffic. One can observe distinctive patterns that are specific to each class of traffic. Crawler traffic tends to exhibit more regularity and stability over time. User traffic, instead, tends to exhibit more local variations. However, over time, user traffic also shows repetitive patterns whose regularity is related to the "human time scale" (e.g., daily or weekly patterns).

To determine the long-term stability of a time series with respect to its quick variations, we leverage autocorrelation analysis techniques. Furthermore, to separate repetitive patterns from slow variations, we leverage decomposition analysis techniques. Decomposition separates a time series into a trend component that captures slow variations, a seasonal component that captures repetitive patterns, and a component that captures the remaining noise. We use the information extracted from these analyses as input to classifiers for detection. These classifiers are used to determine whether an unknown time series belongs to a crawler or a user.

Campaign attribution. Some malicious crawlers willingly reduce their volume of requests to remain stealthy. This comes at a cost for the attacker in terms of crawling efficiency (volume of retrieved data during a specific time span). To compensate for these limitations, malicious crawlers, just like legitimate ones, distribute their crawling activity over multiple sources. We denote a set of crawlers, distributed over multiple locations, and showing synchronized activity, as a *crawling campaign*.

Figure 3 presents two time series that correspond to the same crawler distributed over two hosts in different subnets. One can observe a high level of similarity between the two time series. The key insight of our approach is that these similarities can be used to identify the distributed crawlers that are part of a campaign.

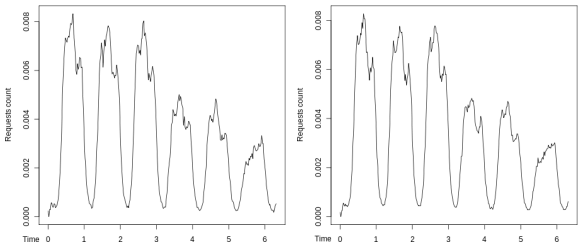


Figure 3: Distributed crawler (sources A and B)

Proactive containment. Our containment approach uses the detection results to establish a whitelist of legitimate crawlers and user sources that are allowed direct access to the site. We also compile a blacklist of unauthorized crawlers that need to be blocked. As mentioned previously, other sources are granted a small number of unrestricted accesses per day. For IPs that exceed this threshold, the system responds by inserting CAPTCHAs or crawler traps into response pages.

3 Crawler Detection Approach

In this section, we provide a more detailed description of the crawler detection process in PUBCRAWL.

3.1 Heuristic detection based on headers

The *heuristic detection* module processes, for each source, the HTTP header fields and URLs extracted from the requests in the traffic log. Request-based features have been used in the past by systems that aim to detect crawlers [10, 11, 17, 18, 21, 22]. We selected the following ones for our detection module:

- *High error rate:* The URL lists used to feed a crawler often contain entries that belong to invalid profile names. As a result, crawlers tend to show a higher rate of errors when accessing profile pages.
- *Low page revisit:* Crawlers tend to avoid revisiting the same pages. Users, on the other hand, tend to regularly revisit the same profiles in their network.
- *Suspicious referrer:* Many crawlers ignore the referrer field in a request, setting it to null instead. Advanced crawlers do handle the referrer field, but give themselves away by using referrers that point to the results of directory queries (listings).
- *Ignored cookies:* Many crawlers ignore cookies. As a result, a new session identifier cookie is issued for each request by these crawlers.

In addition to the previously-described heuristics, we propose a number of additional, novel features:

- *Unbalanced traffic:* When we see multiple user agent strings coming from one IP, we expect the requests to be somewhat balanced between these different user agents. If one agent is responsible for more than 99% of the requests, this is suspicious.
- *Low parameter usage:* Existing detectors mostly consider the length and depth of URLs. In our case, profile URLs show a similar length and the same level of depth. Instead, parameters can additionally be appended to URLs (e.g., language selection). Crawlers typically do not use these parameters.
- *Suspicious profile sequence:* Crawlers often access profiles in a sorted (alphabetic) order. This is because the pointers to profiles are often obtained from directory queries.

Heuristics results are combined into a final classification by a majority vote. That is, if a majority of heuristics are triggered, we flag the source as a crawler.

3.2 Time series extraction

Our traffic shape detection significantly differs from existing work on crawler detection as we do not divide the source traffic into sessions. Instead, we model traffic as counting processes from which some properties are extracted, replacing the simple timing features (e.g., mean and deviation of inter-arrival times) computed over traffic sessions as in [18, 21, 22].

Deriving the time series. We model the traffic from a source over a time window $[t_0, t_n[$ as a counting process (a particular kind of time series). A counting process $X(t) = \{X(t)\}_{t_0}^{t_n}$ counts the number of requests that arrive from a given source within n time intervals of fixed duration, where each interval is a fraction of the entire time window. Notice that the traffic logs must be split into windows of at least two days to capture patterns that repeat at the time scale of one day.

Because most statistical tests are sensitive to the total (absolute) numbers of requests, we normalize the time series. To this end, the amplitude of the series is first scaled to capture the ratio of requests per time interval to the total volume of requests produced by the source. The time frame of the series is then normalized by setting a common time origin: the start of all series is set to be the arrival time of the *first* observed request *among all* monitored sources. Formally, the normalized time series are extracted as follows: Let S be the set of monitored sources. Let R_s be the set of requests from a source $s \in S$. Then, its time series $X_s(t)$ is:

$$X_s(t) = \left\{ \frac{Card(\{r \in R_s \mid r.arrival \in [t, t+d[\})}{Card(\{r \in R_s \mid r.arrival \in [t_0, t_n[\})} \right\} \quad (1)$$

$$t_0 = min_{s \in S}(\{min_{r \in R_s}(\{r.arrival\})\}) \quad (2)$$

We chose d = 30 minutes as the length of each time interval, in order to smooth the shape of the time series. Shorter intervals made the series too sensitive to perturbations in the network communications that are often independent of the source. Longer intervals, instead, make it harder to capture interesting variations in the traffic.

3.3 Detection by traffic shape analysis

In this section, we present how we model the shape of time series to distinguish users from crawlers. Sections 3.3.1 and 3.3.2 discuss how characteristic features of the traffic are extracted using the auto-correlation and decomposition analyses. Section 3.3.3 describes how these features are used to train classifiers with the goal of identifying crawler traffic.

3.3.1 Auto-correlation analysis

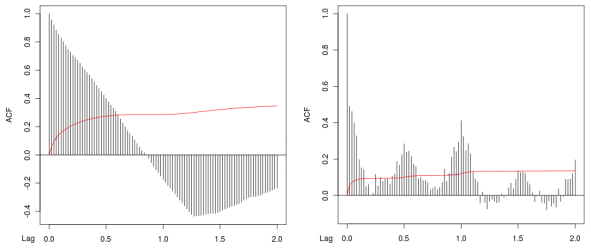
To characterize the stability of the source traffic, we compute the *Sample Auto-Correlation Function (SAC)* of the source's time series and analyze its shape [3, Chapt.9]. The *SAC* captures the dependency of values at different points in time on the values observed for the process at previous times (the time difference between the two compared values is called *lag*). This function is a good indicator for how the request counts vary over time. A strong auto-correlation at small lags indicates a stable (regular) process, which is typical for crawlers. Spikes of auto-correlation at higher lags indicate potential seasonal variations, as in the case of users (for example, a strong auto-correlation at a lag of one day indicates that traffic follows a regular, daily pattern).

For a given lag k, the auto-correlation coefficient r_k is computed as in Equation 3, where E denotes the mean and Var the variance. The *SAC Function* captures the auto-correlation coefficients at different lags.

$$r_k = \frac{E[(X(t) - E[X(t)]) \times (X(t+k) - E[X(t)])]}{Var[X(t)]} \quad (3)$$

To determine the significance of the auto-correlation coefficient at a given lag k, the coefficient is usually compared to the standard error. If the coefficient is larger than twice the standard error, it is statistically significant. In this case, we say that we observe a *spike* at lag k. A spike indicates that counts separated by a lag k are linearly dependent. We use the Moving Average (MA) model to compute the standard error at lag k [3]. Unlike other models, the MA model does not assume that the values of the time series are uncorrelated, random variables. This is important, as we expect request counts from a single source to be correlated.

Figure 4 presents the *SAC Functions* computed over the time series from Figure 2. The functions were plotted over 96 lags (time span of two days). The additional (red) lines correspond to the standard errors under the MA assumption. If we observe the shape of these *SACs*, the crawler *SAC* shows a strong auto-correlation at small lags, followed by a slow linear decay, but no consecutive spike. The user *SAC* shows a less significant auto-correlation at small lags, followed by a fast exponential decay. However, we observe spikes at lags multiple of 0.5, corresponding to a half-daily and daily seasonality.



linear decay, one sign alternation — *exponential decay, local spikes at lags multiple of 0.5*

Figure 4: YahooSlurp and MSIE 7 SACs

Auto-correlation interpretation. Interpreting the shape of the *SAC* is traditionally a manual process, which is left to an analyst [3]. For our system, this process needs to be automated. To this end, we introduce three features to describe the properties of the *SAC*: speed of decay, sign alternation, and local spikes.

The *speed of decay* captures the short-term stability of the traffic. A slow decay indicates that the traffic is stable over longer periods whereas a fast decay indicates that there is little stability. The speed of decay feature can assume four values: linear decay, exponential decay, cut-off decay (coefficients reach a cliff and drop), no decay (coefficients comparable to random noise).

The *sign alternation* identifies how often the sign of the *SAC* changes. Its values are: no alternation, single alternation, oscillation, or erratic. No or single sign alternations are typical of crawlers, while user traffic potentially shows more alternations.

Local spikes reflect periodicity in the traffic. A local spike implies a repeated activity whose occurrences are separated by a fixed time difference (lag). This is typical for user traffic. This feature has two values: a discrete spikes count plus a Boolean value indicating if spikes are observed at interesting lags (half day, day).

Computing our features is a two-step process: First, we compute "runs" over the auto-correlation coefficients of the *SAC*. A run is a sequence of *zeroes* and *ones* for each lag k, where a *one* indicates that a particular property of interest holds. An auto-correlation coefficient is characterized by four properties: positive, significant, null, and larger than the previous value. Runs allow us to determine how often these properties change. This can be done by computing various statistics (e.g. mean, variance, length) over the related runs and their subruns (a sequence of consecutive, identical values).

In the second step, we apply a number of heuristics to the different runs. In particular, we compare the statistics computed for different runs with thresholds that indicate whether a certain property changes once, frequently, or never. The details of how we compute the runs, as well as the heuristics that we apply, are described in detail in Appendix A. The heuristics provide the actual feature values: speed of decay, sign alternation and local spikes.

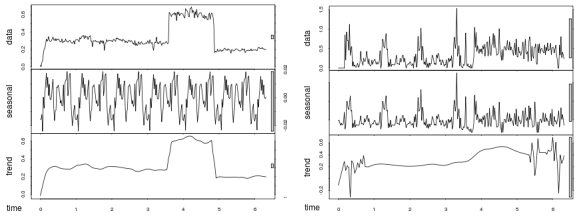


Figure 5: YahooSlurp and MSIE 7 decompositions

3.3.2 Decomposition analysis

The events that constitute a time series are the result of different factors that are not directly captured by the time series; only their impact is observed. Part of these factors slowly change over time. These factors generate slow shifts within the series that constitute its *trend* component $T(t)$. Another part of these factors repeat over time due to periodic events. These factors generate repetitive patterns within the series that constitute its *seasonal* component $S(t)$. Unexplained short-term, random effects constitute the remaining *noise* $R(t)$.

Decomposition aims to identify the three components of a time series such as $X(t) = T(t) + S(t) + R(t)$. The results of decomposition provide valuable insights into which component has the strongest influence on the series. The decomposition is achieved using the *Seasonal-Trend decomposition procedure based on LOESS (STL)*. *STL* moves sliding windows over the analyzed time series to compute smoothed representations of the series using the *locally weighted regression (LOESS)* [6, 7].

The width of the sliding windows is chosen specifically to extract certain frequencies. In our system, we set the window width for both the trend and the seasonal components to 6 hours. This width had to be comparable to the expected seasonality of 12 or 24 hours for user traffic. The shorter width permits the extraction of components that may slightly vary from day to day; a larger width would not tolerate such variation. Figure 5 shows the decomposition of the time series from Figure 2.

Trend variation. The trend components for crawlers are often very stable, and thus, close to a square signal. To distinguish stable from noisy "signals", we apply the differentiation operator ∇ (with a lag 1 and a distance of 1) to the trend. For crawlers where the traffic is rather stable, the differentiated trend is very close to a null series, with the exception of spikes when an amplitude shift occurs. For users, the traffic shows quicker and more frequent variations, which results in a higher variation of the differentiated trend series. The variation of the differentiated trend is measured using the *Index of Dispersion for Counts* [12], $IDC[\nabla T(t)]$. Compared to other statistical indicators such as the variance, the IDC, as a ratio, offers the advantage of being normalized.

Season-to-trend ratio. Time series that correspond to users' traffic often exhibit repetitive patterns. These patterns can repeat on a daily basis, weekly basis, or other frequencies that show some significance in terms of the human perception of time. Consequently, the seasonal component is more important for user traffic and likely prevails over the trend component. This is no longer true for crawler traffic. To capture this difference between user and crawler traffic, we compute the ratio between the seasonal and trend components as shown in Equation 4. The amplitude of the seasonality component is computed using the difference between its maximum and minimum values (as the minimum value might be negative). The amplitude of the trend component is measured using a quantile operation to remove outliers resulting from possible errors in the decomposition.

$$r_{s/t} = \frac{Max[S(t)] - Min[S(t)]}{Quantile[T(t), 0.95]} \quad (4)$$

Table 1: Features characterizing traffic time series

Origin	Feature name	Type
Auto-correlation analysis (SAC)	*coefficients at lags 1 and 2, decay, sign alternation, number of local spikes, daily correlation*	2 x Continuous, 2 x Enumerate(4) 1 x Discrete 1 x Boolean
Decomposition analysis	*differentiated trend IDC, season over trend ratio*	1 x Continuous, 1 x Continuous

3.3.3 Traffic classification

The auto-correlation and decomposition provide a set of features that describe important characteristics related to the traffic from a given source. These features are summarized in Table 1. The type column shows, for each feature, the domain of values: continuous or discrete numbers, Boolean values, or labels drawn from a set of n possibilities (written as "Enumerate(n)").

Of course, no single feature alone is sufficient to unequivocally distinguish between crawler and user traffic. Thus, PUBCRAWL trains a combination of three well-known, simple classifiers. The first classifier is a naive Bayes classifier that was trained using the maximum posterior probability [19]. The conditional probabilities for continuous attributes were estimated using weighted local regression. The second classifier is an association rules classifier [5]. The third classifier is a support vector machine (SVM) that was trained using a non-linear kernel function (Gaussian Radial Basis Function - RBF). To construct an optimal hyperplane, we chose a C-SVM classification error function.

All three classifiers require a training phase. For this, we make use of a labeled training set that contains time series for both known crawlers and users. Each classifier is trained separately on the same training data. During the detection phase, each classifier is invoked in parallel. To determine whether an individual source is a crawler or a user, we use majority voting over the outputs of the three classifiers.

Algorithm 1

Require: A time series set $\chi = X_1(t), ..., X_n(t)$
1: $C = \oslash$
2: **for** $X_i(t)$ in χ **do**
3: $\tau = k/Stdv[X_i(t)]$
4: $C^* = candidates(C, Vol[X_i(t)], Max[X_i(t)], Stdv[X_i(t)])$
5: $s_m, c_m = max_{c \in C^*}(\{euclidean\_dist(X_i(t)), c.medoid)\})$
6: **if** $s_m > \tau$ **then**
7: $c_m.add\_time\_series(X_i(t)$
8: **else**
9: $c_n = new\_cluster(medoid = X_i(t))$
10: $C = C \cup c_n$
11: **end if**
12: **end for**
13: **return** clusters set C

4 Crawling Campaign Attribution

In the previous section, we introduced our approach to detect crawlers based on the analysis of traffic from individual sources. However, numerous crawlers do not operate independently, but work together in crawling campaigns, distributed over multiple locations. By identifying such campaigns, we can provide valuable forensic information to generate the list of blacklisted sources.

Campaign attribution is achieved by identifying crawlers that exhibit a strong similarity between their access patterns. To detect sources that exhibit similar patterns, we use clustering to group similar time series coming from detected crawlers. During the training period, we first determine a minimal intra-cluster similarity required for crawlers to belong to a common campaign. During detection, clustering results are used to identify hosts that exhibit similar activity, and hence, are likely part of a single, distributed crawler.

Time series similarity. A significant amount of literature exists on similarity measures for time series [16]. Most of this body of work aims at providing a similarity measure that is resilient to distortion, so that time series of similar shape can be clustered. Distortion in terms of request volume is already handled by the normalization that we apply when deriving the time series. On the other hand, resilience to time distortion is not desirable. The reason is that we want to detect sources that behave similarly, including the time domain. As a consequence, we leverage the (inverse) *Squared Euclidean Distance*. This metric is fast and known to provide good results [13].

Incremental clustering. PUBCRAWL uses an incremental clustering approach to find similar time series. Time series coming from detected crawlers are submitted one by one to our clustering algorithm described in Algorithm 1. For each new time series, the algorithm computes the similarity between this time series and the medoïds of existing clusters (Line 5, Algorithm 1). When the maximal similarity, found for a given medoïd, is above a threshold τ (Line 6, Algorithm 1), the time series is added to the associated cluster. Otherwise, the time series becomes the medoïd for a new cluster that is created. τ corresponds to the minimal intra-cluster similarity that the algorithm has to enforce. τ is computed during the learning phase, based on a labeled dataset that contains time series for distributed crawlers. Note that τ is not fixed but depends on the standard deviation to compensate for time series proving to be more "noisy".

To speed up the process, for each incoming time series, the function $candidates$ (Line 4, Algorithm 1) selects a subset of comparable clusters. These candidate clusters are chosen because their medoïds have a volume of requests, an amplitude and a standard deviation that are comparable to the new time series. We found that this selection process significantly reduced the number of necessary computations (but also false positives).

Once all time series are processed, each cluster that contains a sufficient number of elements is considered to represent a crawling campaign. Sources that are part of this cluster are flagged as parts of a distributed crawler.

5 Crawler Proactive Containment

Existing techniques to detect crawlers, including our approach, often require a non-negligible amount of traffic before reaching a decision. To address attacks in which an adversary leverages a large number of bots for crawling, we require an additional containment mechanism. In PUBCRAWL, the detection modules are mainly used to produce two lists: A whitelist of IPs corresponding to authorized users (proxies) and legitimate crawlers, and a blacklist of IPs corresponding to suspicious crawlers. These lists are used to enforce an access policy for the real-time stream of requests.

Whenever the protected web site receives a request from a source in the whitelist, the request is granted. Sources on the blacklist are blocked. Other sources are considered unknown, and are treated as follows.

For each source, we check the number of requests that were generated within a given time interval (currently, one day). If this volume remains below a minimal threshold k_1, the source is considered a user, and its access to the site is granted. If k_1 is chosen sufficiently small, the amount of leaked information remains small, even if an attacker has many machines at their disposal.

If this same volume is above a second threshold k_2, we can use our models to make a meaningful decision and either whitelist or blacklist this source.

When the number of requests from a source is between k_1 and k_2, unknown sources are exposed to active responses such as CAPTCHAs and crawler traps. By modifying k_1 and k_2, a number of trade-offs can be made. When k_1 increases, fewer users are exposed to active responses, but it is easier for large-scale, distributed attackers to steal data. When k_2 increases, our system will be more precise in making decisions between crawlers and users but we expose more users to

active responses. In Section 6, we show that, for reasonable settings of the thresholds k_1 and k_2, only a very small fraction of requests and IP addresses are subjected to active responses, while the amount of pages that even large botnets can scrape remains small.

In practice, PUBCRAWL will only be used for "anonymous" requests. These are requests from users who do not have an account on the site or have not logged in. When a user authenticates (logs in), subsequent requests will contain a cookie that grants direct access to the site (and authenticated requests are rate-limited on an individual basis). This is important to consider when discussing why IP addresses form the basis for our white- and blacklists. In fact, we are aware that making decisions based on IP addresses can be challenging; IP addresses are only a weak and imperfect mechanism to identify the source of requests. However, for anonymous requests, the IP address is the only reliable information that is available to the server (since the client completely controls the request content).

One problem with using IP addresses is that a malicious crawler (or bot) on an infected home computer might regularly acquire a new IP address through *dhcp*. Thus, the blacklist can become stale quickly. We address this problem by allowing each individual IP address only a small number k_1 of unrestricted accesses (before active containment is enabled). While each fresh IP address does allow a bot a new set of requests, IP addresses are not changing rapidly enough so that attackers can draw a significant advantage. Another problem is that the IP address of a whitelisted, legitimate proxy could change, subjecting the users behind it to unwanted, active containment. Our experimental results (in Section 6) show that this is typically not the case, and legitimate, high-traffic sources are relatively stable. Finally, it is possible that an attacker compromises a machine behind a whitelisted proxy and abuses it as a crawler. To protect against this case, our system enforces a maximum traffic volume after which the whitelist status is revoked and the IP address is treated as unknown.

To keep the access control lists up-to-date, PUBCRAWL continuously re-evaluates unknown sources and entries on the whitelist. Entries in the blacklist are expired after some days. Moreover, users can always authenticate to the site to bypass PUBCRAWL's checks.

6 Evaluation

We implemented PUBCRAWL in Python, with an interface to R [1] for the time series analysis. The system was developed and evaluated in cooperation with a large, well-known social network. More precisely, we used ten days of real-world request data taken from the web server logs of the social network, and we received feedback from expert practitioners in assessing the quality of our results. The evaluation yielded a favorable outcome, and our detection system is now integrated into the production environment of a large social networking site.

6.1 Dataset presentation

The ten days of web server logs contain all requests to public profile pages of the social networking site we used as a case study. Public profiles are accessible by anyone on the Internet without requiring the client to be logged in. The social network provider has experienced that almost all crawling activity to date comes from clients that are not logged into their system. The reason is that authenticated users are easy to track and to throttle. Handling large volumes of non-authenticated traffic from a single source is most difficult; this traffic might be the result of anonymous users surfing behind a proxy, or it might be the result of crawling. Making this distinction is not straightforward.

The log files contain eight fields for each request: *time, originating IP address, Class-C subnet, user-agent, target URL, server response, referrer, cookie*. The IP address and the Class-C subnet fields were encrypted to avoid privacy issues. Thus, we can only determine whether two requests originate from the same client, or from two clients that are part of the same */24 network*. The remaining information is unmodified. This allows us to check for suspicious user agents, and to determine the profile names that are accessed. The server response can be used to determine whether the visited profile exists or not. In addition, the referrer indicates the previous website visited by the client. The cookie contains, in our case, a session identifier that is set by the social networking site to track individual clients.

Data prefiltering. Given that the social network is very popular, the log files are large – they contain tens of millions of requests per day that originate from millions of different IP addresses. As a result, we introduce a prefiltering process to reduce the data to a volume that is manageable by our time series analyses. To this end, we leverage the fact that, by looking at the volume of requests from a single source, certain clients can be immediately discarded: we can safely classify all sources that generate more than 500,000 requests per day as crawlers.

Sources that generate less than 1,000 requests per day are also put aside because our time-series-based techniques require a minimum number of data points to produce statistically meaningful results. These sources are handled by the active containment policy.

In our experiments, the prefiltering process reduced the number of requests that need to be analyzed by a factor of two. More importantly, however, the number of clients (IP addresses) that need to be considered is reduced by about four orders of magnitude.

Ground truth. Obtaining ground truth for any real-world data is difficult. We followed a semi-automated approach to establish the ground truth for our datasets.

For the initial labels, we used heuristic detection (Section 3.1), which represents the state-of-the-art in crawler detection. We then contacted the social network site, which had access to the actual (non-encrypted) IP addresses. Based on their feedback, we made readjustments to the initial ground truth labels. More precisely, we first marked sources as legitimate crawlers when they operated from IP ranges associated with popular search engines. In addition, IP addresses that belong to well-known companies were labeled as users. For borderline cases, if an IP address was originating traffic from users who successfully logged on to the site, we tagged this IP as a user. Overall, we observed that current heuristics often incorrectly classify high volume sources as crawlers.

In a next step, we performed an extensive manual analysis of the sources (by looking at the time series, checking user agent string values, ...). We made a second set of adjustments to the ground truth. In particular, we found a number of crawlers that were missed by heuristic detection. These crawlers were actively attempting to mimic the behavior of a browser: user agent strings from known web browsers, cookie and referrer management, and slow runs at night. Some examples of mimicry are discussed in Appendix B. These cases are very interesting because they underline the need for more robust crawler detection approaches such as PUBCRAWL.

Finally, we manually checked for similar time series, and correlated this similarity with user agent strings and class-C subnets. This information was used to build a reference clustering to evaluate the campaign attribution.

6.2 Training detection and attribution

For training, we used a first dataset S_0, which contained five days worth of traffic recorded between the 1^{st} and the 5^{th} of August 2011. After prefiltering, this dataset consists of $\sim$73 million requests generated by 813 IP addresses. Given this number of IP addresses, manual investigation for the ground truth was possible.

Heuristic detection. We used the training set to individually determine suitable thresholds for the detection heuristics. We verified the results of the configured heuristics over the training set S_0 with the ground truth. The results are given in Table 2. In this table, a true positive (TP) means a correctly identified crawler. A true negative (TN) is a correctly identified user. The results are split between heuristics over features from existing work and new features introduced in this paper. We can see that the new features greatly improve the detection rate when combined with existing ones. Still, the final detection rate of 75.54% illustrated the need for more robust features.

Table 2: Training: Accuracy for heuristic detection.

Rates	Former features	New features	Combined features
TP/FN	41.32%/58.68%	82.31%/17.69%	**75.54%/24.46%**
TN/FP	100.00%/0.00%	84.00%/16.00%	**96.00%/04.00%**

Table 3: Training: Accuracy for traffic shape detection.

Crawlers (TP/FN)	Bayes	Rules	SVM	**Vote**
Cross validation	98.39%	96.36%	98.55%	**98.99%/01.01%**
Two third split	97.45%	96.19%	95.11%	**96.90%/03.10%**
Users (TN/FP)	Bayes	Rules	SVM	**Vote**
Cross validation	79.09%	78.91%	81.91%	**82.91%/17.09%**
Two third split	78.84%	78.22%	80.44%	**82.28%/17.72%**

Table 4: Training: Accuracy for campaign attribution.

Precision	Recall	Accuracy
99.03%	85.54%	**94.35**%

Traffic shape detection. To train the classifiers for detection, we used S_0 that contained 709 crawler sources and 104 user sources. To determine the quality of the training over S_0, we used both five-fold cross validation and a validation by splitting the data into two thirds for training and one third for testing. The results are shown in Table 3. The table shows that our system obtains a crawler detection rate above 96.9%. It also shows the benefit of voting, as the final output of classification is more accurate than each individual classifier.

Interestingly, the dataset was not evenly balanced between crawlers and users. The majority of sources that produce more than thousand requests per day are crawlers. However, the dataset also contains a non-trivial amount of user sources. Thus, it is not possible to simply block all IP addresses that send more than one thousand requests. In fact, since the user sources are typically proxies for a large user population, blocking these nodes would be particularly problematic. We thus verified the accuracy specifically for user sources in Table 3. Given the bias towards crawlers, the accuracy for users is slightly lower but remains at around 83%.

Note that traffic shape detection results show interesting improvements compared to heuristic detection that is close to the approach deployed in existing work. This approach produces more accurate results while using features that are more robust to evasion.

Campaign attribution. For campaign attribution, the clustering algorithm presented in Section 4 needs to be configured with a τ that defines the minimal, desired similarity within clusters. To determine this threshold, we ran a bisection clustering algorithm on the training dataset S_0. The algorithm first assumes that all elements (time series) are part of a single, large cluster. Then, it iteratively splits clusters until each time series is part of a single cluster. We analyzed the entire cluster hierarchy to find the reference clusters as well as the necessary cut points to obtain them. The cut points of reference clusters indicated us the minimal similarity that we related to the deviation to determine that

$k = 350$ was the linear coefficient giving optimal values for τ. For the candidate selection, the following thresholds were chosen: $volume = 25\%$, $amplitude = 35\%$, $deviation = 30\%$, so that these thresholds avoid additional false positives while creating no false negatives compared to the results without candidate selection.

To evaluate the quality of the campaign attribution, we ran our clustering algorithm on the 709 crawler sources from S_0. We use the *precision* to measure how well the clustering algorithm distinguished between time series that are different, and the *recall* to measure how well our technique recognizes similar time series. To evaluate the successful attribution rate, we use the *accuracy* to measure how well the clustering results can be used to detect distributed crawlers and thus campaigns.

The clustering results are shown in Table 4. One can see that the algorithm offers a good precision. The recall is a bit lower: Closer examination revealed that a few large reference clusters were split into multiple clusters. For example, *Bingbot* had its 209 corresponding time series split into 5 subclusters. Fortunately, recall is less important in our attribution scenario. The reason is that split clusters do not prevent us to detect a campaign in most cases; instead, a campaign is simply split into several smaller ones.

6.3 Evaluating detection capabilities

For testing, the social networking site provided an additional dataset S_1, which contained five extra days worth of traffic. After prefiltering, this dataset consisted of $\sim$62 million requests generated by 763 IP addresses. Unlike the training, the testing was performed on site at the social networking site. Hence, the traffic logs could be analyzed with non-encrypted IPs.

Heuristic detection. We compared the results for the heuristic detection over the testing set S_1 with the ground truth we had (semi-automatically) established previously. As shown in Table 5, the detection rate slightly decreases to 71.60%, but remains comparable to the rate over the training set.

Traffic shape detection. To test the classifiers trained over S_0, we deployed the traffic shape detection approach over the test set S_1. The results for this experiment are presented in Table 6. According to the table, the global accuracy of 94.89% remains very close to the 95% of accuracy observed for the training set.

Since the goal of the detection module is to build whitelists and blacklists of sources, we computed individual results for the following four subsets: users (5%) and legitimate crawlers (65%) to be whitelisted, and unauthorized crawlers (7%) and masquerading crawlers (23%) to be blacklisted. Unauthorized crawlers are agents that can be recognized by their user agent string but are not supposed to crawl the site. Masquerading crawlers are malicious crawlers trying to masquerade as real browsers to remain stealthy and to avoid detection.

Table 5: Testing: Accuracy for heuristic detection.

Rates	Former features	New features	Combined features
TP/FN	31.19%/68.81%	86.24%/14.76%	**71.60%/28.40%**
TN/FP	100.00%/0.00%	87.18%/12.82%	**94.87%/05.13%**

Table 6: Testing: Accuracy for traffic shape detection.

	Bayes	Rules	SVM	Vote
Global	93.05%	87.55%	94.36%	**94.89%**
Legitimate crawlers	92.54%	87.10%	97.18%	**93.95%**
Unauthorized crawlers	88.89%	96.27%	100.00%	**100.00%**
Masquerading crawlers	98.27%	86.71%	98.84%	**98.84%**
Crawlers (TP/FN)	93.66%	87.68%	97.79%	**95.58%/04.42%**
Users (TN/FP)	82.50%	85.00%	32.50%	**82.50%/17.50%**

Table 7: Testing: Accuracy for campaign attribution.

Precision	Recall	Accuracy
92.84%	80.63%	**91.89%**

We achieve perfect detection for unauthorized crawlers. In particular, the system was able to detect crawlers such as *ISA connectivity checker*, *Yandexbot*, *YooiBot*, or *Exabot*. Results are also very good for masquerading browsers, with a detection rate close to 99%. The detection rate slightly drops to 94% for legitimate crawlers such as *Baiduspider*, *Bingbot*, *Googlebot* or *Yahooslurp*. But 4% of these false negatives are related to *Google FeedFetcher*. In principle, the requests from *FeedFetcher* are triggered by user requests. As a result, its time series are individually recognized as user traffic.

By combining heuristic detection and traffic shape detection, the detection rates were not improved further. The reason is that the crawlers detected by heuristics were already included in the set of crawlers detected by traffic shape. This observations confirms our belief that traffic shape detection is stronger than heuristic detection based on HTML and URL features.

To gain a better understanding of our results, we asked for feedback from the social networking site. The network administrators confirmed that a large number of crawlers were previously unknown to them (and they subsequently white- or blacklisted the IPs). Since deanonymized IP addresses were available to us, we could check the sources of these crawlers. Interestingly, several sources were proxies of universities, where crawler traffic was mixed with user activity. Because of the mix of user and crawler activity, the current detection techniques did not raise alerts. Note that such mix of activity must be taken into consideration for blacklisting (e.g., the university administrators can be warned that unauthorized crawling is coming from their network and asked to take appropriate measure). In such cases, it is a policy decision whether to blacklist the IP or not. Also, recall that requests from users who are logged-in is not affected.

Performance. To process the entire dataset S_1, several instances of the modules for heuristic detection, traffic shape detection and campaign attribution were run over five parallel processes that required roughly 45 minutes to run. This time included loading the data into the database, generating the time series, and performing the different analyses. The experiments were run on a single server (with 4 cores and 24 GB of RAM).

6.4 Evaluation of campaign attribution

To evaluate the quality of the campaign attribution technique, we ran our clustering algorithm over the crawler sources from the de-anonymized testing set S_1. The results of the experiment are shown in Table 7. One can see that the precision and recall have slightly dropped compared to the training set. The intra-cluster similarity threshold τ might not be optimal anymore. Nonetheless, the attribution accuracy remains at 91.89%, which is close to the 94% obtained during training.

Overall, we obtain 238 clusters from the 763 distinct source IPs. Looking at these clusters, we started to observe interesting campaigns when a cluster contained 3-4 or more elements (hosts). Table 8 provides a description of these campaigns. The first campaigns correspond to legitimate crawlers. Interestingly, the campaign associated to *Feedfetcher*, whose crawlers evaded traffic shape detection, is successfully identified. Even if the *Feedfetcher* traffic is similar to user traffic, *Google* distributes the requests (i.e., load-balances) over multiple IPs, explaining that their time series are synchronized. Table 8 also presents five campaigns showing suspicious user-agent strings, and five campaigns masquerading as legitimate browsers or search engine spiders. Another interesting case is the campaign where crawlers send requests as *Gecko/20100101 FireFox*. This campaign shows a significant number of clusters because it uses *rotating IP addresses* over short time periods. However, it is still detected because the active IP addresses were operating loosely synchronized. The social network operators showed a particular interest in this case, and they are now relying on our system to detect such threats that are difficult to detect otherwise.

6.5 Evaluation of proactive containment

The evaluation was performed over a dataset S_2 of non-filtered traffic. The dataset contained $\sim$110 million requests from $\sim$11 million IP sources.

Figure 6 shows the cumulative distribution function (CDF) for the fraction of IPs (y-axis) that send (at most) a certain number of requests (x-axis). One can see that most IPs (more than 98.7%) send only a single request per day. This is important because it means that most sources (IPs) will never be affected by active containment. Figure 7 shows the situation for requests instead of IPs. That is, the figure shows the CDF for the fraction of total requests (y-axis) over the maximum number of requests per source (x-axis). One can see that roughly 45% of all request are sent by sources that send at most 100 requests. The graph highlights that a little over 40% of all requests are done by sources that make only a single request. One can also see that a significant amount of the total requests are incurred by a few heavy hitter IPs that make tens of thousands to millions of requests.

Table 8: Identified Crawling Campaigns.

Agent	#Clust.	#ClassC	#IP	Req/day
Legitimate crawlers				
Bingbot	5	11	211	6 million
Googlebot	5	2	42	4 million
+ Feedfetcher	4	4	65	–
Yahooslurp	4	9	71	500 thousand
Baiduspider	1	1	23	50 thousand
Voilabot	3	3	20	19 thousand
Facebookexternalhit/1.1	1	1	8	14 thousand
Crawlers with suspicious agent strings				
" "	2	16	22	330 thousand
Python-urllib/1.17	2	51	54	140 thousand
Mozilla(compatible;ICS)	1	10	10	70 thousand
EventMachine HTTP Client	1	3	3	3 thousand
Gogospider	1	2	3	2 thousand
Masquerading crawlers				
Gecko/200805906 FireFox	1	10	73	350 thousand
Gecko/20100101 FireFox	9	12	25	60 thousand
MSIE6 NT5.2 TencentTraveler	1	1	30	7 thousand
Mozilla(compatible; Mac OS X)	1	1	4	8 thousand
googlebot(crawl@google.com)	1	1	4	1 thousand

Containment impact. We use these two graphs to discuss the impact of the two containment thresholds: k_1, below which sources have unrestricted access to the site, and k_2, above which sources can be examined by our analysis (and hence, properly whitelisted or blacklisted).

We can see from Figure 6 that if we choose k_1 low enough, we can guarantee that only a tiny number of sources will be impacted. For example, by setting $k_1 = 100$, we see that 99.98% of the sources will not be impacted at all. If an attacker wants to take advantage of this unrestricted access, he would require 5,000 crawlers running in parallel to reach the crawling rate of a *single* crawler agent from *Googlebot*. Looking back at Table 8 for real-world examples, the lowest rate of requests we observed for a distributed crawler was for the *Gecko/20100101 FireFox* campaign using rotating IPs. Even in this case, the amount of requests per agent was above a few hundreds per day. Thus, we consider values for k_1 between 10 and 100.

To choose k_2, we have a trade-off between the amount of traffic that will be impacted by active responses and the quality of our detection. We see the fraction of requests that are impacted by plotting k_1 and k_2 over Figure 7 and reading the difference over the y-axis. Table 9 lists the proportion of impacted traffic for various concrete settings of k_1 and k_2. If we keep k_2 at 1,000 and choose 20 for k_1, we expect active responses to impact less than 0.1% of all IP sources **and** only about 3.24%

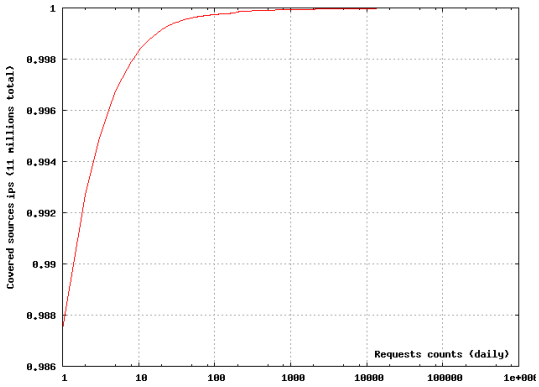


Figure 6: CDF of the source IPs over traffic volumes

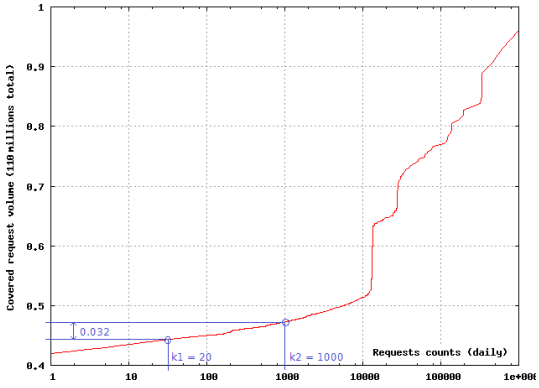


Figure 7: CDF of the requests over traffic volumes

Table 9: Requests (%) impacted by containment

k_1/k_2	100	500	1000	2000	5000	10000
10	1.49%	2.95%	3.75%	4.82%	6.27%	7.88%
15	1.20%	2.66%	3.46%	4.53%	5.98%	7.58%
20	0.99%	2.44%	**3.24**%	4.32%	5.77%	7.37%
25	0.81%	2.27%	3.07%	4.14%	5.59%	7.19%
50	0.37%	1.83%	2.63%	3.70%	5.15%	6.75%
100	0.00%	1.46%	2.26%	3.33%	4.78%	6.38%

of the overall requests. Of course, we expect that these 3.24% of requests do contain a non-trivial amount of traffic from stealthy crawlers. When k_1 is increased, the impact on legitimate users decreases. The downside is that large botnets can scrape larger parts of the site.

Sources stability. The whitelist approach for legitimate, high-volume sources (over k_2) works well only when IP sources remain stable for user proxies and legitimate crawlers. To verify this assumption, we studied the IP evolution between the training set S_0 and the testing set S_1. Considering crawlers, 66.9% of IPs were both present in S_0 and S_1. Looking at the stable IPs, they correspond either to legitimate crawlers (e.g., *Googlebot*, *Bingbot*) or large crawling campaigns (e.g., *Python*, *Firefox* from Table 8). Unstable crawler IPs correspond to unauthorized and masquerading crawlers.

Most importantly, looking at high-volume users (proxies), 81.8% of IPs were both present in S_0 and S_1 (about one month apart). Thus, whitelisting these sources would work well and, hence, we would not see much negative effect for (non-authenticated) users when using active containment. Moreover, the 18.2% of non-stable IPs are mainly due to sources whose volume was close to the prefiltering threshold. These sources might have been stripped from S_0 or S_1 by prefiltering.

Overall, our results demonstrate that PUBCRAWL can thwart large-scale malicious crawlers and botnets while interfering with a small number of legitimate requests.

7 Limitations

Our system has proven to be useful in the real-world, and it is currently deployed by the social networking site as a part of their crawler mitigation efforts. However, as with many other areas in security, there is no silver bullet, and sophisticated attackers might try to bypass our system. Nevertheless, PUBCRAWL significantly increases the difficulty bar for the attackers.

Detection limitations. An attacker might try to thwart the heuristics-based and traffic-shape-based detection modules. The traffic shape detection has two main requirements: *1)* a large-enough volume of requests is required for the time series to be statistically significant, *2)* at least two days of traffic are required for the auto-correlation and decomposition analyses.

While traffic shape detection is well-suited for detecting crawlers of sufficient volume, because of requirement *1)*, it is not particularly well-suited to detect "slow" distributed crawlers that send a very small number of requests from hundreds of thousands of hosts. For this, campaign attribution is more appropriate. Because of requirement *2)*, it is not well suited either to detect "aggressive" (noisy) crawlers in real-time. For this, heuristic detection is more appropriate.

To address the problem of "slow" and "aggressive" crawlers, PUBCRAWL combines the detection modules with a containment strategy. Aggressive crawlers are initially slowed down by active responses, until they are detected and blacklisted. Slow crawlers can at most make k_1 requests before the active response component is activated. Moreover, since slow crawlers require a larger number of machines, the effect of the active response component is magnified (applied to each crawler).

Another way to avoid detection is traffic reshaping. That is, an attacker could try to engineer the crawler traffic so that it closely mimics the behavior of actual users. The attacker would first have *1)* to craft valid HTTP traffic (headers) and *2)* to design a stealthy visiting policy both in terms of *topology* and *timing*. In terms of topology, the attacker would have to craft a non-suspicious sequence of URLs to visit (based on order and revisit behavior). In terms of timing, he would have to craft the volume and distribution of requests over time so that the traffic shape remains similar to user traffic. Overall,

mimicking the behavior of users would require a non-trivial effort on behalf of the attacker to learn the properties of user traffic, especially given that only the social network has a global overview of the incoming traffic.

Attribution limitations. If an attacker wants to bypass our campaign attribution method, he has to ensure that *all* nodes of his distributed crawler behave differently. Sets of rotating IPs are already successfully detected. To break the synchronization between the nodes, simple time shifts would not be sufficient: Existing similarity measures for time series (e.g., dynamic time warping) can be used to recover from shifts. An attacker would have to completely break the synchronization between its different crawlers while shaping for each one a different traffic behavior (which needs to be similar to user traffic to avoid individual detection).

Containment limitations. If attackers do not succeed in whitelisting their crawlers, they can willingly maintain their traffic volume under the containment threshold k_1. To crawl a good portion of a social network with millions of pages requires a significant crawling infrastructure, and building or renting botnets over long periods of time might be prohibitively expensive for most adversaries. Attackers can also increase their traffic volume until the blocking threshold k_2. In this case, they would have to find a solution to automatically bypass active responses (resolve CATPCHAs or identify crawler traps).

8 Related Work

We are not the first one to study the problem of detecting web crawlers. However, we are the first to propose a solution to distributed crawlers, and we are the first to have used an extensive, large-scale real-world dataset to evaluate and validate our approach.

Similar to [18, 21, 22], PUBCRAWL relies on machine learning techniques to extract characteristic properties of traffic that can help to distinguish crawlers from users. However, the features we use for the learning process are different. Compared to [11, 17, 21, 22], the similar features we extract from the HTTP headers and URLs are fed to heuristic detection. Our experiments demonstrate that these features are not reliable.

For traffic shape detection and its learning process, we used timing features instead. Compared to [18, 21, 22], the results of the auto-correlation and decomposition analyses prove to be more robust. That is, the extracted properties are harder to evade by attackers than the simple time and volume statistics used by previous approaches (e.g., the average or the variance of inter-arrival times between requests).

In our detection approach, most of the features extracted from the time series are designed to express the regularity of web traffic. In [8], the authors already leveraged the notion of regularity of crawler traffic for detection. To extract the relevant information, the authors rely on Fast Fourier Transformations. However, both crawler and user traffics show regularities, but they do so at different levels. We thus use decomposition techniques to distinguish between different types of regularities: Crawler regularity can be observed within the trend component, whereas user regularity can be observed within the seasonal component.

Existing crawler detection approaches mainly remain deployed offline – just like our detection approach based on traffic shape. However, a significant novelty of our approach is that we integrate our detection process into a proactive containment strategy to protect from crawlers in real-time. This is similar to [18] where the authors address real-time containment. The containment approach they propose relies on the detection results from an incremental classification system where crawler models evolve over time. Instead, we chose a more realistic, practical white- and blacklisting approach where sources are blocked on a per IP basis.

An interesting contribution of our work is the formalization of the traffic by time series, which allows us to address the problem of crawling campaign attribution and distributed crawlers detection using clustering. We have also evaluated our tool on a much larger scale than previous work, which has used requests that are in the order of thousands. Finally, in our experiments, we observed and identified real-world evasion techniques that target some of the traffic features used in previous work. Hence, we provide evidence that attackers today are investing significant effort to evade some of the straightforward and well-known crawler detection techniques.

9 Conclusion

To defend against malicious crawling activities, many websites deploy heuristics-based techniques. As a reaction, crawlers have increased in sophistication. They now frequently mimic the behavior of browsers, as well as distribute their activity over multiple hosts.

This paper introduced PUBCRAWL, a novel approach for the detection and containment of crawlers. The key insight of our system is that the traffic shape of crawlers and users differ significantly so that they can be automatically identified using time series analysis. We also propose the first technique that is able to identify crawling campaigns by detecting the synchronized activity of distributed crawlers using time series clustering. Finally, we introduce an active containment mechanism that strategically uses active responses to maximize protection and minimize user annoyance.

Our experiments with a large, popular social networking site demonstrate that PUBCRAWL can distinguish users with accuracy and filter out crawlers. Our detec-

tion approach is currently deployed in production by the social networking site we collaborated with.

Acknowledgment

We would like to thank the people working at the social network with whom we collaborated, as well as Secure Business Austria for their support. This work was supported by the Office of Naval Research (ONR) under Grant N000140911042 and the National Science Foundation (NSF) under grants CNS-0845559, CNS-0905537 and CNS-1116777.

References

[1] The R Project for Statistical Computing. http://www.r-project.org/.

[2] 67th District Court, Tarrant County, Texas. Cause NO. 067-194022-02: American Airlines, Inc. vs. FareChase, Inc. http://www.fornova.net/documents/AAFareChase.pdf, 2003.

[3] B. L. Bowerman, R. T. O'Connell, and A. B. Koehler. *Forecasting, Time Series, and Regression – An applied approach – Fourth edition*. Thomson Brook/Cole, 2005.

[4] E. Burzstein, R. Bauxis, H. Paskov, D. Perito, C. Fabry, and J. Mitchell. The Failure of Noise-Based Non-Continuous Audio Captchas. In *IEEE Security and Privacy*, 2011.

[5] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[6] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. STL: a Seasonal-Trend Decomposition Procedure based on Loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[7] W. S. Cleveland and S. J. Devlin. Locally Weighted Regression: an Approach to Regression Analysis by Local Fitting. *J. Am. Stat. Assoc*, 83:596–610, 1988.

[8] M. D. Dikaiakosa, A. Stassopouloub, and L. Papageorgioua. An Investigation of Web Crawler Behavior: Characterization and Metrics. *Computer Networks*, 28:880–897, 2005.

[9] D. Doran and S. S. Gokhale. Discovering New Trends in Web Robot Traffic through Functional Classification. In *Proc. of the IEEE International Symposium on Networking Computing and Applications (NCA)*, pages 275–278, 2008.

[10] D. Doran and S. S. Gokhale. Web Robot Detection Techniques: Overview and Limitations. *Data Mining and Knowledge Discovery*, 22(1-2):183–210, 2011.

[11] W. Guo, S. Ju, and Y. Gu. Web Robot Detection Techniques based on Statistics of their Requested URL Resources. In *Proc. of the International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 302–306, 2005.

[12] R. Gusella. Characterizing the Variability of Arrival Processes with Indexes of Dispersion. *"IEEE J. Sel. Areas Commun.*, 9(2):203–211, 1991.

[13] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: a Survey and Empirical Demonstration. In *Proc. of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 102–111, 2002.

[14] M. Koster. A Method for Web Robots Control. Technical report, RFC draft, 1996.

[15] M. Lamberton, E. Levy-Abegnoli, and P. Thubert. System and Method for Enabling a Web Site Robot Trap. United States Patent No. US 6,925,465 B2, 2005.

[16] T. W. Liao. Clustering of Time Series Data – a Survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[17] X. Lin, L. Quan, and H. Wu. An Automatic Scheme to Categorize User Sessions in Modern HTTP Traffic. In *Proc. of GLOBECOM*, pages 1485–1490, 2008.

[18] A. G. Lourenço and O. O. Belo. Applying Clickstream Data Mining to Real-Time Web Crawler Detection and Containment using ClickTips Platform. In *Advances in Data Analysis, Proc. of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V.*, pages 351–358. Springer, 2007.

[19] A. McCallum and K. Nigam. A Comparison of Event Models for Naive Bayes Text Classification. In *AAAI FICML Workshop on Learning for Text Categorization*, 1998.

[20] Pinsent Masons. Ryanair wins German court victory in screen-scraping injunction. http://www.theregister.co.uk/2008/07/11/ryanair_screen_scraping_victory/, 2008.

[21] A. Stassopoulou and M. D. Dikaiakos. Crawler detection: A bayesian approach. In *Proc. of the International Conference on Internet Surveillance and Protection (ICISP)*, 2006.

[22] P.-N. Tan and V. Kumar. Discovery of Web Robot Sessions based on their Navigational Patterns. *Data Mining and Knowledge Discovery*, 6(1):9–35, 2002.

[23] Tech Crunch. Hacker arrested for blackmailing StudiVZ and other social networks. http://eu.techcrunch.com/2009/10/21/hacker-arrested-for-blackmailing-studivz-and-other-social-networks/, 2009.

[24] L. von Ahn, M. Blum, N. Hopper, and J. Langford. The CAPTCHA Project. Technical report, Carnegie Mellon University, 2000.

[25] P. Warden. How I got sued by Facebook. http://petewarden.typepad.com/searchbrowser/2010/04/how-i-got-sued-by-facebook.html, 2010.

A Interpreting auto-correlation functions

This section describes how we compute the values of the three features describing the shape of the *Sample Auto-Correlation Function (SAC)*. We first compute *runs* over four *SAC* characteristics: trend (increase/decrease), sign (positive/negative), significance (spikes), and null (lags with null correlation). The *run* computations are shown in Equations 5-8 for the auto-correlation coefficient r_k that ranges over all k lags.

$$\text{Trend: } tr_k = \begin{cases} 1 & \text{if } k = 1 \text{ or } |r_k| \geq |r_{k-1}| \\ 0 & \text{else} \end{cases} \quad (5)$$

$$\text{Sign: } sr_k = \begin{cases} 1 & \text{if } r_k \geq 0 \\ 0 & \text{else} \end{cases} \quad (6)$$

$$\text{Significance: } pr_k = \begin{cases} 1 & \text{if } |r_k| \geq 2 \times s_k \\ 0 & \text{else} \end{cases} \quad (7)$$

$$\text{Null: } nr_k = \begin{cases} 1 & \text{if } |r_k| \leq 0.1 \times max\{|r_k|\} \\ 0 & \text{else} \end{cases} \quad (8)$$

Based on the previously-computed runs (as well as amplitude values), we determine the three properties of the *SAC*: Table 10 lists the heuristics to determine the decay, Table 11 lists the heuristics to determine the sign alternation, and Table 12 lists the heuristics to identify local spikes.

B Crawlers mimicking user behavior

Table 13 shows examples of crawlers that we found in our dataset. The first example in the table represents an easy crawler to detect: The user-agent points to the *Python* programming framework, which is popular among crawlers, no

Table 10: Heuristics to determine the speed of decay of the SAC

Characteristic	Metric	Expected Value	Interpretation	Feature Value
Amplitude	difference between first and last lags	low	no overall decrease in the amplitude of the coefficients	*No decay (white noise)*
Trend runs	mean of the runs values	average	balanced proportion between increases and decreases	
	number of runs	high	high number of inflections points in the function	
Null runs	mean of the runs values	low	observed coefficients are not residuals	
Amplitude	difference between first and middle lags	above average	quick decrease in the amplitude of the coefficients	*Cut-off*
Spike runs	length of the first run	low	coefficients become insignificant after a short number of lags	
	length of the longest null valued run	high	the function contains long intervals of insignificant coefficients	
Null runs	mean of the runs values	above average	the function coefficients tend quickly towards 0	
	lag of the first positive run	low	the function converges towards 0 in the small lags	
Amplitude	difference between first and middle lags	below average	slow decrease in the amplitude of the coefficients	*Linear decay*
Trend runs	number of runs	low	low number of inflections points in the function	
Spike runs	length of the first run	above low	coefficients remain significant after a longer number of lags	
Spike runs	length of the first run	above low	coefficients remain significant after a longer number of lags	
Null runs	mean of the runs values	low	the function coefficients tend slowly towards 0	
-	-		all decaying functions that are neither cut off nor linear	*Exponential decay*

Table 11: Heuristics to determine the sign alternation property of the SAC

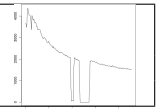
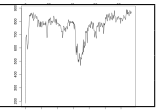
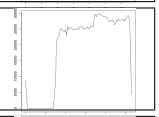
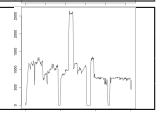
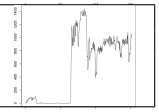
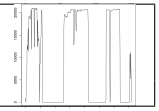
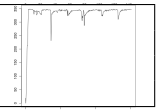
Characteristic	Metric	Expected Value	Interpretation	Feature Value
Sign runs	number of runs	equal 1	no sign change	*No alternation*
Sign runs	number of runs	equal 2	single sign change	*Single alternation*
Sign runs	number of runs	above average	important number of sign changes	*Oscillations*
	variance of the runs length	below average	sign changes occur at regular periods	
-	-		sign changes are unpredictable	*Erratic alternation*

Table 12: Heuristics to determine the local spikes of the SAC

Characteristic	Metric	Expected Values	Interpretation	Feature Value
Spike runs	lags of positive runs	above 1	first spike must be ignored	*Local spike (lag, length)*
	length of positive runs	low	short spikes centered around a lag	

referrer is set, the cookies transmitted by the server are ignored, the dispersion index is low, indicating regular traffic, and the error rate is too high to be generated by a user following links. The next examples introduce different evasion techniques by mimicry. Examples 2 to 7 use user-agent string masquerading. Examples 4 and 5 set their referrer to the result of a directory query, whereas Example 7 uses mainly profiles URL as referrers, just like during human browsing. Examples 6 to 8 handle cookies with different policies: Examples 6 and 7 reuse the same cookie for a fixed number of requests, whereas Example 8 reuses the same cookie for a fixed period of time. More advanced techniques of traffic shaping can also be found, where the timing and the targeting of requests is modified to look similar to user traffic. Example 6 interrupts its activity at night to avoid raising suspicion. Examples 6 and 7 show a high fraction of revisited pages, which is more typical for human behavior. These examples indicate that crawlers already attempt to bypass existing detection heuristics.

Table 13: Evasion techniques observed for different crawlers.

Source	Referrer	Cookie	IDC[X(t)]	Overlap	Errors	Traffic
Grey cells correspond to observed properties of the crawler traffic that are close to browser traffic, they might correspond to evasion attempts.						
IP: B2ED1877DE4DC83B3DF0ED8AFF68E6B0490B5107 User-agent string: Python-urllib/1.17	null referrer	no cookie reuse	517.79	00.0%	12.7%	
IP: 8DB9FE9C5D0796FF7E9BFB05201EE55A9052C586 User-agent string: Mozilla/4.0 (Windows; U; Windows NT 6.1; en-US; rv...)	null referrer	no cookie reuse	10.49	01.3%	01.4%	
IP: A30B14FA5261FA04C612F5E938FCB349C28CBF58 User-agent string: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)	null referrer	no cookie reuse	7502.97	00.0%	00.4%	
IP: 43DB44548E1AB95696703B92DBBD778CB44EC4E4 User-agent string: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)	directory query as referrer (all starting with same letter)	no cookie reuse	246.12	00.2%	00.2%	
IP: 62A7DF089C1700ACF7A2B1A1614418E97C4ED42B User-agent string: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV...)	directory query as referrer (all starting with same letter)	no cookie reuse	430.32	00.1%	00.3%	
IP: 3E680C6B9A92070AB608671486DC332AD2B7083F User-agent string: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1)	null referrer	reuse cookie for next 3 requests	9669.75	11.7%	03.3%	
IP: 2DC89E853294C1F0797002C4211FE005E39BEA52 User-agent string: Mozilla/4.0 (compatible; MSIE 5.0b2; Windows NT...)	only 1.6% of null referrers, rest is all public profiles	reuse cookie for next 100 requests	4.35	29.3%	00.3%	
IP: 3CBEF4B3A90AA47F6517260D78371EDBFE09E9A7 User-agent string: ia_archiver (+http://www.alexa.com/site/help/webma...)	null referrer	reuse cookies for requests in a 15 min interval	59.42	02.7%	01.2%