# Detecting Stealthy, Distributed SSH Brute-Forcing

Mobin Javed[†] and Vern Paxson[†◇]

[†]University of California, Berkeley [◇]International Computer Science Institute

## Abstract

In this work we propose a general approach for detecting distributed malicious activity in which individual attack sources each operate in a stealthy, low-profile manner. We base our approach on observing statistically significant changes in a parameter that summarizes *aggregate* activity, bracketing a distributed attack in time, and then determining which sources present during that interval appear to have coordinated their activity. We apply this approach to the problem of detecting stealthy distributed SSH bruteforcing activity, showing that we can model the process of legitimate users failing to authenticate using a beta-binomial distribution, which enables us to tune a detector that trades off an expected level of false positives versus time-to-detection. Using the detector we study the prevalence of distributed bruteforcing, finding dozens of instances in an extensive 8-year dataset collected from a site with several thousand SSH users. Many of the attacks—some of which last months—would be quite difficult to detect individually. While a number of the attacks reflect indiscriminant global probing, we also find attacks that targeted only the local site, as well as occasional attacks that succeeded.

## Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS—*Security and Protection*

## Keywords

Scanning; SSH; Brute-forcing; Distributed

## 1. INTRODUCTION

A longstanding challenge for detecting malicious activity has been the problem of how to identify attacks spread across numerous sources, such that the individual activity of any given source remains modest, and thus potentially not particularly out of the ordinary. These scenarios can arise whenever a detector employs a *threshold* used to flag that a given candidate attack source has exhibited a suspiciously high level of activity (e.g., when conducting scanning or DoS flooding). Attackers can respond to such detection procedures by employing multiple sources in order to thin out their activity to prevent any single source from exceeding the threshold; their attack becomes *distributed* and therefore potentially *stealthy*, i.e., hard to detect based on any individualized analysis.

In this work we present a general strategy for potentially detecting such stealthy activity, which consists of two basic steps. First, we employ the statistical technique of *change-point detection* to identify times during which a global property has shifted—indicating that, *in aggregate*, a site's activity reflects the presence of problematic activity. We then determine the range of time over which this activity occurred and, within that interval, identify which sources appear to have contributed to the activity.

In particular, we apply this approach to the problem of detecting *distributed SSH brute-forcing*: attackers employing a number of systems that each try different username/password combinations against a site's SSH login servers, hoping that one of them will stumble across a working combination made possible by a careless user. The threat of SSH brute-forcing is well-known: indeed, any SSH server open to general Internet access receives incessant probing by hostile remote systems that energetically attempt to locate instances of weak authentication [5]. The degree to which such attempts also occur in a stealthy slow-but-steady fashion, however, has attracted little study. The difference between single energetic probes and stealthy distributed ones is significant: defenders can easily detect the former, and therefore either block the activity or investigate it (to ensure none of the attempts succeeded). The latter, however, poses a much more difficult detection problem. If each host in a distributed brute-forcing attack itself only attempts username/password logins at a low rate, then distinguishing hostile activity from the inevitable login failures made by legitimate user errors becomes much more difficult. Yet the distinction is vital: a pattern of *attempt*/*attempt*/*attempt*/*success* made by a legitimate user simply reflects a set of typos, or a password that took a few stabs to remember; but by a distributed SSH brute-forcer, it provides the only slender indication of success amongst a mass of probing that in aggregate predominantly failed.

We aim to both provide an exemplar of our general strategy in terms of detecting distributed (but *coordinated*) SSH brute-forcing attacks, as well as developing an assessment of the prevalence of such attacks as seen over years of data. In terms of our two-step approach, we first identify *attack epochs* during which in aggregate we can with statistical confidence determine that some sort of SSH brute-forcing event occurred. Here, we employ change-point detection framed in terms of a parameter that summarizes the network/server activity of groups of remote hosts—in particular, the *aggregate login failure rate*. Our second step classifies the hosts appearing during the detected epochs as either *participants* or *non-participants* in the activity, based on both individual past his-


*CCS'13,* November 4–8, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2508859.2516719.

tory and "coordination glue", i.e., the degree to which a given host manifests patterns of probing similar to that of other hosts during the epoch.

We develop and evaluate our detector on 8 years of SSH login records collected via central syslogging at the Lawrence Berkeley National Laboratory, a large ($\approx$ 4,000 employees and visitors) research facility. We measure and quantify the duration, intensity and behavior of the detected attacks. We find multiple large-scale coordinated attacks from botnets, the longest one spanning about 1.5 months. All the attacks we detect would have been completely missed by a point-wise host-based detector. We correlate these attacks with data from several other sources, finding that half of the large-scale incidents at the site are part of global attacks, with a significant average overlap of $\approx 70\%$ attack hosts appearing at multiple sites in the same time span.

We organize the rest of the paper as follows. We begin with related work in § 2. § 3 details the characteristics of the dataset we use in developing and evaluating our detector. § 4 frames our detection approach. In § 5 we develop a model of the process by which legitimate users make authentication errors when attempting to log in, which serves as the basis for parameterizing our SSH password brute-force detector. We discuss our evaluation results and findings in § 6, and summarize in § 7.

## 2. RELATED WORK

The literature relevant to our work lies in three domains: (i) coordinated attack detection, (ii) SSH brute-force attack detection, and (iii) studies of the prevalence of SSH brute-forcing activity.

The detection of coordinated attacks has received little treatment in the literature. The earliest work of which we are aware is that of Staniford et al., who correlate anomalous events using simulated annealing for clustering [17]. Gate's work on coordinated scan detection is the most prominent subsequent effort in this domain [8]. Given an input set of scan sources, Gate's algorithm extracts the subset of hosts that appear coordinated by using a set-covering approach; the premise is that the attacker divides the work among the coordinating scanning hosts in a manner that maximizes information gain while minimizing work overlap. For our purposes this work has two limitations: (i) the individual attack hosts require pointwise identification, and thus the approach will not find stealthy attacks, and (ii) the algorithm lacks a procedure for determining when a site is under attack. Other work has addressed the somewhat similar problem of detecting DDoS attacks, but these detection approaches face a difficult problem of how to differentiate attack participants from legitimate users [19, 16].

With regard to SSH brute-forcing, host-based detection techniques such as DenyHosts [2], BlockHosts [1], BruteForceBlocker [9], fail2ban [12], and sshguard [3] block hosts that cross a threshold for failed attempts in a specified amount of time. Other work has developed network-based approaches. Kumagai et al. propose an increase in the number of DNS PTR record queries to detect SSH dictionary attacks [13]. This increase results from the SSH server logging the fully qualified domain names of the SSH clients attempting access. This work does not discuss how to establish detection thresholds, nor does it present an evaluation of the system's accuracy. Vykopal et al. develop flow signatures for SSH dictionary attacks [18]. They show that a large number of short flows having a few bytes transferred in both directions and appearing together in a short duration of time are indicative of failed login attempts, providing the means to then detect brute-force attacks from flow data. Hellemons also studied the possibility of using only flow data to detect SSH brute-force attacks, modeling the brute-force attacks as consisting of three phases: scanning, brute-force and die-off (in case of successful compromise) [11]. They monitor the ranges of three parameters—flows-per-second, packets-per-flow and bytes-per-packet—to identify these phases. Both of these works test their detectors only on simulated dictionary attacks, and do not address how to distinguish instances of forgotten usernames/passwords from brute-forcers. More generally, none of these brute-force detection approaches have the ability to detect stealthy coordinated attacks.

Malecot et al. use information visualization techniques to detect distributed SSH brute-force attacks [14]. For each local host, the remote IP addresses that attempt to log in are displayed using a quadtree—a tree data structure formed by recursively subdividing two dimensional space into four quadrants. The procedure performs 16 iterations to map 32-bit IP addresses onto a quadtree, each time deciding the next sub-quadrant by looking at the next two bits of the IP address. The analyst then visually compares quadtrees for different local hosts to identify as coordinated attackers remote IP address(es) that appear in quadtrees of multiple local hosts.

Finally, regarding the prevalence of SSH brute-force attacks, Bezut et al. studied four months of SSH brute-force data collected using three honeypot machines [6]. They find recurring brute-forcing activity, sometimes with several weeks in between, indicating that the attacks target a wide range of IP address space. Owens et al. performed a measurement study of SSH brute-force attacks by analyzing data from honeypots on three networks—a small business network, a residential network, and a university network—for eleven weeks during 2007–2008 [15]. They find that the number of login attempts during different attacks varied from 1 or 2 to thousands. More than a third of the attacks consisted of ten or fewer login attempts. They find instances of both slow and distributed attacks designed to evade detection. They also find that precompiled lists of usernames and passwords are shared across different attackers, identifying five such dictionaries. Their study reveals that only 11% of the attempted passwords are dictionary words.

## 3. DATASETS AND DATA FILTERING

We evaluate our detector on eight years of SSH login data collected at the Lawrence Berkeley National Laboratory (LBNL), a US national research laboratory. The temporal breadth of this dataset allows us to study attack patterns at the site across the years. We also draw upon SSH datasets from four other sites spread across the IP address space (and several geographic locations) to assess whether attacks we detect at LBNL reflect targeted behavior or indiscriminant probing. We refer to these sites as HONEY, RSRCHLAB, HOMEOFF, and CAMPOFF, and describe them below. In this section we present these datasets and discuss ways in which we filtered the data for our subsequent analysis.

### 3.1 Main dataset

Table 1 provides summary statistics for our main dataset, LBNL. This site's systems primarily reside in two /16 address blocks (from two separate /8's). Only a small fraction of the address space runs externally accessible SSH servers, providing access to both individual user machines and compute clusters. The benign SSH activity in this data consists of interactive as well as scripted logins.

For this site we have datasets collected at two vantage points: (i) logs collected by a central syslog server that records information about login attempts reported by (most of) the SSH servers, and (ii) flow data for SSH port 22 collected by border monitoring. For each login attempt, the syslog data provides the time, client

| Time span | Jan 2005–Dec 2012 |
|---|---|
| SSH servers | 2,243 |
| Valid users | 4,364 |
| Distinct valid user/server pairs | 10,809 |
| Login attempts | 12,917,223 |
| Login successes | 8,935,298 |
| Remote clients | 154,318 |
| Attempts using passwords | 5,354,833 |
| successes | 1,416,590 |
| remote clients | 119,826 |
| SSH border flows | 215,244,481 |
| remote clients seen in flows | 140,164 |
| High-rate brute-forcers | 7,476 |
| Mean attempts per high-rate brute-forcer | 382.84 |
| Mean daily password login attempts | 486.13 ($\sigma = 182.95$) |
| Mean daily users | 116.44 ($\sigma = 32.41$) |

Table 1: Summary of LBNL syslog and flow data.

and server[1] IP addresses, username on the server, whether the login succeeded, and the authentication type used. The flow data supplements this perspective by providing contact information (but no details) for attempts to access IP addresses that do not run an SSH server, or that run an SSH server that does not log via the central syslog server. This data thus enables us to establish the complete[2] set of machines targeted by an attack.

**Filtering.** For the central syslog data, we work with the subset of SSH authentication types vulnerable to brute-forcing (i.e., we omit those using public key authentication), about half of the attempts. We perform all of the characterizations and analyses in the remainder of the paper in terms of this subset.

In addition, we filter this dataset to remove individual brute-forcers that we can readily detect using a per-host threshold for the number of failed login attempts per remote host within a window of time. Given the ease of detecting these brute-forcers, they do not reflect an interesting problem for our detector, though they would heavily dominate the data by sheer volume if we kept them as part of our analysis.

To identify and remove such brute-forcers, we need to empirically establish reasonable thresholds for such a per-host detector. We do so by analyzing the process by which legitimate users make password authentication failures, as follows. We assume that any user who makes repeated failed login attempts followed by a successful attempt reflects a legitimate user. (This assumption may allow consideration of a few successful SSH brute-forcers as "legitimate", but these are very rare and thus will not skew the results.)

Figure 1 plots the number of failed attempts such users make prior to finally succeeding. We see that instances exceeding 10 failed attempts are quite rare, but do happen occasionally. Accordingly, we consider 20 failed attempts as constituting a conservative threshold. We manually analyzed the instances of legitimate users falling after this cutoff (the upper right tail in the figure) and found they all reflect apparent misconfigurations where the user evidently set up automation but misconfigured the associated password. Thus, we deem any client exhibiting 20 or more failures logging into a single server (with no success) over a one-hour period with as a *high-rate brute-forcer*, and remove the client's entire

[1] Some of the syslog records log the server's hostname rather than its IP address. For these we correlated the records against the site's DNS and DHCP logs to resolve to IP addresses.

[2] The flow data has some gaps, though we have it in full for each attack we identified. These gaps are the source of observing fewer "remote clients seen in flows" than "Remote clients" in Table 1.

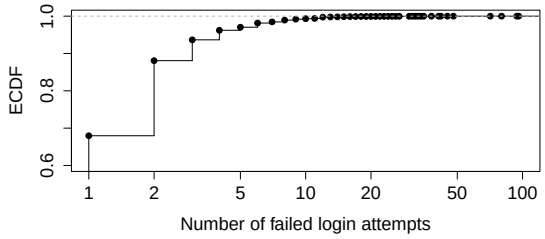


Figure 1: Empirical CDF of the number of failed login attempts per hour until a success for legitimate user login efforts with forgotten or mistyped usernames/passwords.

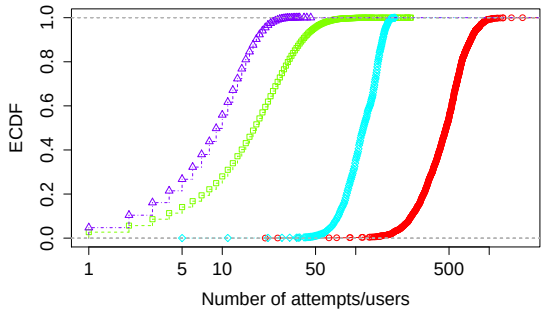


Figure 2: Empirical CDFs for benign password-based SSH usage in LBNL data. Left to right: (i) valid users per hour, (ii) successful logins per hour, (iii) valid users per day, (iv) successful attempts per day.

activity from our dataset. Table 1 summarizes the brute-forcers removed using this definition.

Finally, to give a sense of the volume of activity that remains after these filtering steps, Figure 2 shows the empirical CDF of the hourly and daily numbers of successful logins. A typical day sees 500 successful logins (maximum seen: 1,200) involving 117 distinct users (maximum: 197).

## 3.2 Correlation datasets

The HONEY dataset reflects five manually identified SSH brute-forcing "campaigns" (our term for ongoing instances, as discussed later) as captured by 2 SSH honeypot servers in Norway [4]. Table 2 summarizes these campaigns, which for the most part we characterize as large-scale, persistent, and stealthy. For all but the last campaign, many of the remote clients would evade detection by our simple per-host detector.

The RSRCHLAB dataset reflects flow data from the International Computer Science Institute, a research organization in Berkeley, CA, with a /23 address block. The dataset spans the same time as that of LBNL, though due to the limitations of flow data, we cannot

| Attack Episode | Days | Remote clients | Login attempts | Avg. attempts per remote client |
|---|---|---|---|---|
| Oct 2009–Jan 2010 | 78 | 4,158 | 44,513 | 10 ($\sigma$=24) |
| Jun 2010–Aug 2010 | 56 | 5,568 | 23,009 | 4 ($\sigma$=7) |
| Oct 2011 | 6 | 338 | 4,773 | 14 ($\sigma$=16) |
| Nov 2011 | 13 | 252 | 4,903 | 20 ($\sigma$=24) |
| Apr 2012 | 6 | 23 | 4,757 | 206 ($\sigma$=760) |

Table 2: Summary of attacks in the HONEY data.

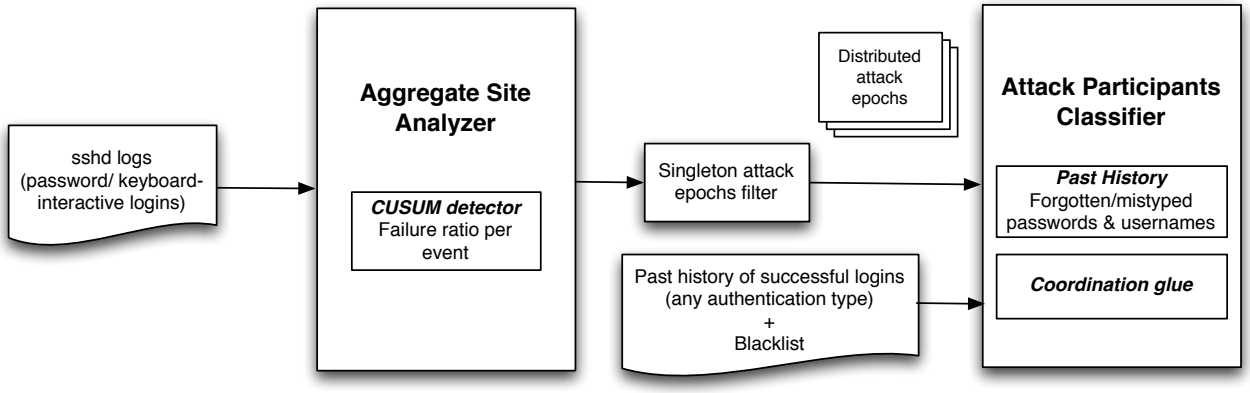


Figure 3: System diagram of our distributed SSH brute-forcing detector

establish how many coordinated attacks exist in it. We discuss our correlation strategy for this dataset in § 6.

HOMEOFF and CAMPOFF capture inbound SSH flow data for a home office (HOMEOFF) and a campus office (CAMPOFF), both in Cleveland, OH (but in separate netblocks). The data likewise spans Jan 2005–Dec 2012, and again due to its limitation to flow data, we cannot identify coordinated attacks in these datasets.

# 4. DETECTION

We structure our detection approach as the sequential application of two components. In general terms, the *Aggregate Site Analyzer* first monitors the site's activity to detect *when* an attack of some sort occurs. Upon detection, the *Attack Participants Classifier* analyzes the activity to identify *who* participated in the attack (which remote systems). In this section we develop both the general detection framework and our particular implementation of it for detecting and analyzing distributed SSH brute-forcing attacks. Figure 3 presents a system diagram for this latter specific context.

## 4.1 Aggregate Site Analyzer

This component detects the general presence of a coordinated, distributed attack based on complete view of a site's activity. We based the approach on devising a *site-wide* parameter that aggregates information from individual users/machines/events at the site. The *Aggregate Site Analyzer* monitors the probability distribution of this parameter for excessive change and flags the deviations as attacks. Often a single party can by itself induce significant change in the parameter. In this case (which arises for detecting SSH brute-forcing) we need to employ a filtering step to omit *singleton* attacks from the alarms, if our overall goal is focused on detecting distributed activity.

One important design question concerns the *accumulation granularity* of the site-wide parameter, i.e., over what sort of collection of activity do we compute it. For example, if the parameter relates to arrival rates, then windows of time will often be natural choices. In other scenarios, we might achieve better results by defining a normalized parameter amenable to longitudinal comparison. To illustrate, consider the scenario where the two consecutive time windows have 50 and 4 attempts, respectively. If these also have 25 and 2 problematic attempts, then we would equate them as having equivalent failure rates; but if we expect failures to appear somewhat rarely, we would like to treat the first instance as much more striking than the second, as the latter only needs some modest degree of stochastic fluctuation to occur. Thus, comparing *ratios* across time can prove misleading.

Given that, for detectors that monitor failure ratios we can benefit from a different accumulation granularity: if we compute the site-wide parameter in terms of *events*—defined as the occurrence of $n$ attempts—then we needn't worry about the effects of stochastic fluctuation that can manifest when using windows of time. In addition, such time-variant events have the property that they allow for faster detection of high-rate attacks, as the detector does not need to wait a fixed amount of time before detecting the attack.

In the case of detecting distributed SSH brute-force attacks, we can define an apt *site-wide* parameter, Global Failure Indicator (GFI) as:

$$\text{GFI} = \text{Number of failed login attempts per event}$$

where an event occurs every $n$ login attempts to the site. These $n$ attempts represent a collection of users authenticating to the site, where the number of users will generally vary across events. (Note, while we could normalize GFI by dividing by $n$, we leave it as a count rather than a ratio, since our subsequent modeling benefits from using discrete variables.) We show in § 5 that in the absence of an attack (i.e., when failures only occur due to mistakes by legitimate users), this distribution is well-modeled as beta-binomial.

Brute-force activity perturbs the GFI distribution, shifting its mean to a higher value. We use sequential change-point detection to detect significant increases in the mean GFI. In comparison to threshold-based detection, sequential change-point schemes can detect small incremental effects that cumulatively lead to an eventual change of mean. This property enables the detector to detect stealthy attacks. Specifically, we use a Cumulative Sum (CUSUM) change-detection algorithm, as prior work has shown its sensitivity to small shifts in the mean [10].

**CUSUM Algorithm.** CUSUM models significant changes as shifts in the mean of a random variable from negative to positive. To use it requires transforming an original random variable $Y$ to an associated value $Z$ that has a negative mean under normal operation. One achieves this by subtracting the empirical mean $\mu$ of $Y$ plus a small reference value $k$, i.e., $Z_n = Y_n - \mu - k$. To do so we need to compute $\mu$ based on data that describes normal operation (no attack underway); see § 5 for how we identify such activity. Finally, note that with little modification to the general framework we can for convenience select $k$ so that $Z_n$ is integer-valued rather

than real-valued. We do so for our subsequent development of the detector.

We then accumulate $Z_n$ over time using the following (again discrete) test statistic: $S_n = \max(0, S_{n-1} + Z_n)$, where $S_0 = 0$. In the case of no change, the value of $S_n$ hovers around zero, but in the face of a change (increase), $S_n$ starts to accumulate in the positive direction.

By convention, one terms the situation of the mean corresponding to normality as *in-control*. When the mean shifts by an amount $\Delta\mu$, one terms the situation *out-of-control*, which corresponds to an attack in our problem domain. Note that the choice of $\Delta\mu$ is specific to the problem we design the detector to detect. In some situations, we might desire a small $\Delta\mu$, while in others we might only have interest in detecting changes corresponding to a large value of $\Delta\mu$. In practical terms, we achieve a given target $\Delta\mu$ by setting two detector parameters, $k$ and $H$, as discussed below.

The algorithm flags an *out-of-control* situation when $S_n$ crosses an operator-set threshold, $H$. The subsequent appearance of an event with normal mean marks the return of the situation to *in-control*, and we reset the test statistic $S_n$ to zero at this point. Thus, the CUSUM detector decides whether the mean has shifted or not according to the decision function $D_n$:

$$D_n = \begin{cases} 1, & \text{if } S_n > S_{n-1} \text{ and } S_n > H \\ 0, & \text{otherwise.} \end{cases}$$

**Determining CUSUM parameters and span of change.** One tunes the parameters $k$ and $H$ of CUSUM based on: the amount of change $\Delta\mu$ to detect, the desired false alarm rate, and the desired time to detection. First, a general rule of thumb when designing a CUSUM detector to detect a mean shift of $\Delta\mu$ is to set $k$ equal to half the change in shift [10]. The other parameter, $H$, controls both the false alarm rate and the detection speed. A lower $H$ means faster detection but a higher false alarm rate.

To assess the balance between these two, we consider the effects of $H$ on the average number of steps the CUSUM detector takes to raise an alarm under in-control and out-of-control distributions. (Note that the first of these corresponds to alarms reflecting false positives, while the latter corresponds to true positives.) We refer to these as *in-control ARL* (Average Run Length) and *out-of-control ARL*, respectively, and choose the value of $H$ that results in the closest match with the desired ARLs.

To determine these ARLs, we can model the CUSUM process as a Markov chain with finite states $X_0, X_1, \ldots, X_H$, corresponding to the test statistic values $S_n \leq 0, S_n = 1, S_n = 2, \ldots, S_n \geq H$ respectively. (Recall that we constrain $Z$ and thus $S$ to discrete integer values.) Note that $X_H$ is the absorbing state. The transition probabilities of this Markov chain depend only on the underlying distribution of the random variable $Z$:

$$P[X_i \to X_0] = P[Z \leq -i]$$
$$P[X_i \to X_j] = P[Z = j - i]$$
$$P[X_i \to X_H] = P[Z \geq H - i]$$

For the intuition behind this formulation, consider the first equation. If the cumulative sum has reached $i$ (i.e., $S_n = i$, corresponding to the state $X_i$) then the possible ways for progressing from it to the state $X_0$ (i.e., $S_n \leq 0$) are to add a value of $Z$ less than or equal to $-i$. A similar perspective holds for the other two equations. Given the complete transition probability matrix $\mathbf{R}$ of the Markov chain, we can compute the above probabilities and the *in-control ARL* as:

$$\textit{in-control ARL} = (\mathbf{I} - \mathbf{R})^{-1}\mathbf{1}$$

where $\mathbf{R}$ is the transition probability matrix, $\mathbf{I}$ is the $(H+1) \times (H+1)$ identity matrix, and $\mathbf{1}$ the $(H+1) \times 1$ matrix of ones [7].

We can likewise compute the *out-of-control ARL* of the detector using the same formulation but substituting $k' = k - \Delta\mu$ [10]. We can then estimate the point of the true start of a change by subtracting the value of *out-of-control ARL* (detection delay) from the time of detection.

Finally, the *Aggregate Site Analyzer* reports the information from CUSUM in the form of *attack epochs*. An attack epoch constitutes of: (i) the set of consecutive *out-of-control* events (i.e., $i = 1 \ldots n$ where $D_i = 1$), and (ii) the set of previous events also incorporated into the epoch based on stepping back through the number of events given by the *out-of-control ARL*.

Each attack epoch can reflect instances of either *singleton* or *coordinated* attacks. The first of these corresponds to a global perturbation of the site-wide variable $Y$ induced by a single source. The second refers to the perturbation arising due to the combined action of multiple sources. Since in this work we focus on distributed attack epochs, we need at this point to exclude singleton attacks.[3] We do so by checking whether CUSUM still flags any events in the epoch as reflecting an attack even if we remove the remote host with the highest number of failed login attempts. If so, we mark the attack epoch as a coordinated attack epoch, and proceed to the second component of our analysis. Otherwise, we discard the epoch as uninteresting (which occurred about 3/4s of the time).

## 4.2 Attack Participants Classifier

The second component of our general detection approach addresses how to go from the global site-wide view to that of individual entities. Here we employ a set of heuristics to analyze activity in the attack epochs flagged by the *Aggregate Site Analyzer* to identify *who* participated in the attack. (The need for heuristics rather than more principled identification arises almost fundamentally from the problem domain: if we could directly identify participants with confidence, we could very likely use the same approach to develop an effective pointwise detector and not have to employ a separate approach for detecting stealthy distributed activity in the first place.)

For our particular problem of detecting distributed SSH brute-force attacks, the individual entities we wish to identify are remote hosts (clients). In addition to the problem of including remote hosts corresponding to legitimate users within it, a distributed attack epoch—particularly if spanning a long period of time—can capture multiple brute-forcers, some of whom might operate in a coordinated fashion, while others might function independently. For example, an attack epoch we detect that includes activity from five remote hosts might in fact be composed of four coordinating remote hosts and one singleton brute-forcer that happens to probe the site at the same time.

For each remote host that appears during the attack epoch, we make a decision about whether to classify it as a legitimate remote host, a singleton brute-forcer (operating alone), or a brute-forcer working with other hosts as part of a coordinated attack. This decision might require manual analysis, as sometimes the categories have significant overlap. To illustrate, Figure 4 diagrams the characteristics that remote hosts in each of these categories can manifest. Legitimate users that fail due to forgotten or mistyped usernames/passwords generally exhibit only a modest number of attempts, similar to low-rate distributed brute-forcers. A remote

[3] Note that such single sources can arise even though we previously filtered out high-rate brute-forcers (per § 3.1) because these singletons might spread their activity across multiple servers, or probe at a rate lower than the 20 failures/hour threshold.

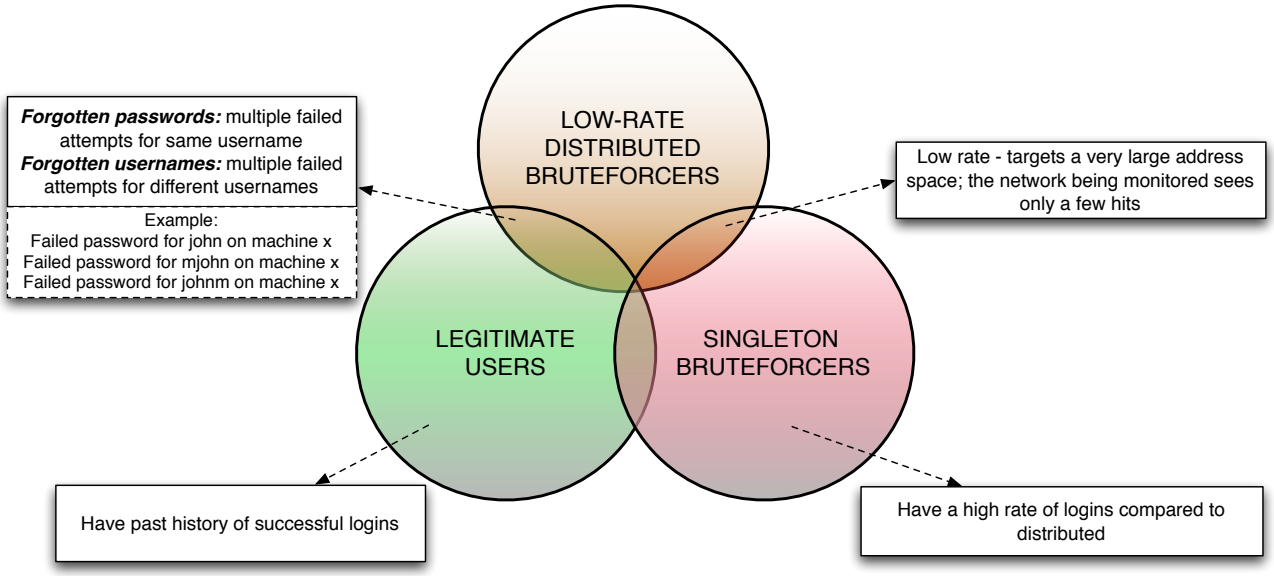


Figure 4: Possible characteristics of remote hosts that fail.

client with no past history of successful logins (see below) provides us with little indication as to whether it reflects a legitimate user or a distributed brute-forcer. Likewise, while we can readily identify singleton brute-forcers that probe at rates higher than distributed brute-forcers, ones that probe at low rates fall into a grey area that we find difficult to automatically classify.

Our classification procedure has two parts. First, we make a set of decisions based on past history. Second, for the remaining hosts during an attack epoch we assess the degree to which they evince the same *coordination glue*: that is, commonality in the set of servers and/or usernames that the hosts probe. The premise behind this second step comes from assuming that attack hosts in a distributed attack aim to work together to achieve a particular task: attackers achieve little utility if their multiple attack hosts do not effectively focus their work. We might also expect that coordinated attack hosts probe at similar levels (number of attempts) due to use of common automation software.

**Classifying activity based on past history.** To facilitate our discussion of this step, we use the following notation: L refers to a *Local host*; R to a *Remote host*; and U to a *Username*. Given that, we classify activity by analyzing past history as follows:

(1) *Forgotten/mistyped passwords:* we identify $\langle$R, U$\rangle$ pairs that have authenticated successfully in the past to any local machine at the site, and consider such activity benign, removing it from the attack epoch. (We filter $\langle$R, U$\rangle$ pairs instead of only remote hosts because multiple users might be behind a NAT.) We can safely filter out this set because the remote client has already established its ability to successfully authenticate as the given user, and thus has no need to try to guess that user's password. While the current activity is not necessarily benign (due to the possibility of a malicious login using stolen/compromised credentials), that scenario lies outside the scope of what we try to detect here.

(2) *Forgotten/mistyped usernames:* the previous step of filtering $\langle$R, U$\rangle$ pairs will miss instances of benign failures when the user mistypes or fails to recall their username. To identify such failures, for each remote host we determine whether it produced a successful login in the past to *any* internal host using a username *closely related* (edit distance = 1) to a username present in the event. If so, we again mark the $\langle$R, U$\rangle$ pair as benign. (Note that we skip this step if we have previously identified $R$ as a singleton brute-forcer.)

**Identifying coordination glue.** After filtering out some activity on the basis of past history, we then turn to making decisions about attack participation amongst the remaining activity on the basis of evidence of "coordination glue", as discussed above. We expect to find either a common set-of-local-servers or set-of-usernames as the coordination glue in most of attacks.

We identify such glue using an approach based on bi-clustering. We construct a bipartite graph with the remote clients as the first node set A and either usernames or local servers as the second node set B. We create an edge between remote client $r$ in node set A and a username $u$ (local server $l$) in node set B if $r$ made a login attempt as $u$ (to $l$). For each graph we then look for partitions of the graph that maximize the edges within the partitions and exhibit very few edges across partitions. We mark nodes belonging to very small partitions as either legitimate users or coincidental singleton brute-forcers, and remove them.

Presently we proceed with the above as a manual process, which we find tractable since the number of attack epochs that emerge from our various filtering steps is quite manageable. In addition, we observe that a distributed attack that targets many hosts on the Internet might lead to activity at the site that exhibits little in the way of apparent coordination glue. In these scenarios, we also take into account timing patterns, number of attempts per remote host, and the presence of an alphabetical progression of usernames as alternate forms of coordination glue.

## 5. MODELING USER AUTHENTICATION FAILURES

To apply the approach developed in the previous section, we need to model the process by which legitimate users make authentication errors when attempting to log in. We want to achieve this modeling not simply in a highly aggregated fashion (e.g., by determining a global benign-user failure rate), but distributionally, as the latter will allow us to form well-grounded estimates of the expected false positives and time-to-detection behavior of our detection procedure. In particular, capturing the distribution will allow us to model GFI (number of failures per event of $n$ login attempts), and thus to determine the Markov chain probabilities required to compute average-run-lengths.

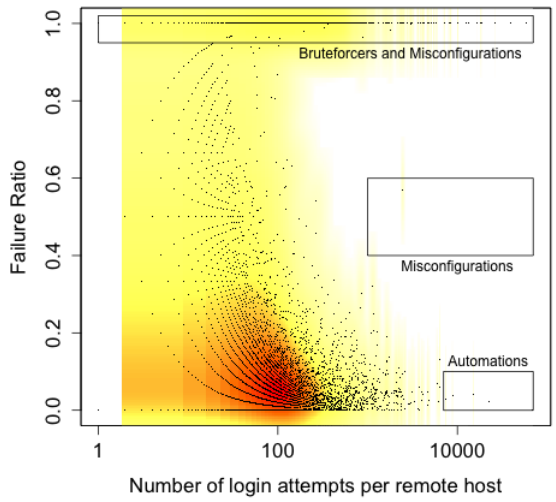


Figure 5: Number of logins and failure ratio of remote hosts over the complete dataset. Note that the dataset has been filtered to remove high-rate brute-forcers that can be detected pointwise using per host detection.

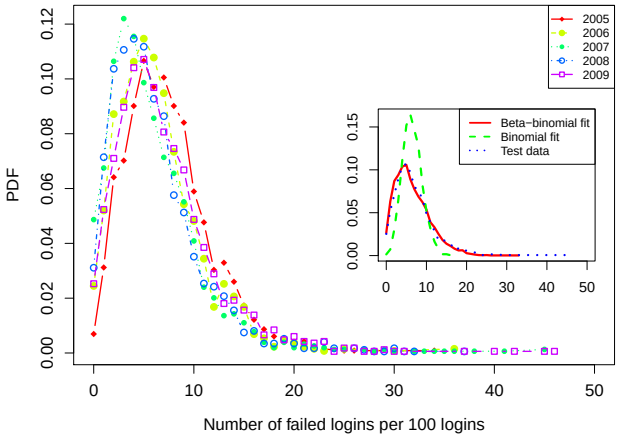


Figure 6: Probability distribution of GFI with n=100 logins.

In order to extract a characterization of legitimate login failures from the LBNL syslog data, we need to first identify clients within it that do *not* reflect legitimate users. Figure 5 shows a heat map of the number of login attempts vs. failure ratio per remote host computed over the complete data for password-based authentication (except with high-rate brute-forcers already filtered out, per § 3). The major density of remote clients resides in the lower left and middle areas of the plot; these mainly depict benign activity. The top region of the plot is highly dominated by brute-forcers (those that went slowly or broadly enough to survive the high-rate filtering), with a few legitimate users in the top-left corner, and some possible misconfigurations throughout. The mid-right region, with a high number of login attempts and a failure ratio in the range 0.4–0.6, very likely consists of misconfigured automations. Finally, the lower-right region captures well-configured automations; these use scripts that log in repeatedly with the correct password, and thus no chance of failure, except for intervening interactive access.

We now discuss our techniques to clean the data to remove both brute-forcers and automations (both well-configured and misconfigured) in the data. After doing so, we illustrate a distribution that fits the cleaned data quite well, providing us with the means to then model GFI.

**Removing brute-forcers.** Accurately identifying low-rate brute-forcers poses a circular problem, as that is exactly what we ultimately set out to detect in this work. Instead we develop an *approximate* procedure to remove the brute-forcers, as follows. We remove from the dataset all remote hosts that never achieve a successful login attempt. The chance that this removes legitimate hosts is low, because it should be rare that a legitimate user repeatedly attempts to log in and never succeeds.

This heuristic removes all of the brute-forcers except the successful ones. In order to cull them, we remove remote hosts with failure ratio $\geq$ 0.95 and $\geq$ 20 login attempts. We choose these thresholds based on our earlier finding that legitimate users extremely rarely have 20 failed login attempts before success (Figure 1). We find 13 remote hosts whose activity falls within these thresholds. Our manual investigation of these determined that six of them reflect misconfigurations (established due to periodicity in the attempts), six eluded classification (activity low enough that they could reflect either legitimate users or participants in coordinated attacks), and one brute-forcer that succeeded in breaking in (clearly an attacker due to employment of a dictionary of generic usernames).

**Removing automations and misconfigurations.** To find candidates for automated activity, we used Zhang's $\chi^2$-based detection [20], which tests for uniformity of when activity occurs in terms of seconds-of-the-minute and minutes-of-the-hour. The premise of this approach is that human-initiated activity should be well-modeled as uniform in terms of these fine-grained timing elements, but automation will generally sharply deviate due to the use of periodic execution.

We applied the test on 3-day windows of activity, requiring each remote client to have at least 50 logins during the window. (We chose the parameters as reflecting clear automation even if spread out over a good amount of time). We used a significance level of 0.001 and a two-sided test in order to detect both non-uniform and extremely-uniform distributions, as both of these likely correspond to automated activity.

The test flagged 363 instances of remote hosts. We manually assessed the 79 of these that triggered detection in multiple windows, since these present instances of long-term automations that can skew our modeling. Doing so found 9 remote hosts that engaged in well-configured long-term automation, and 5 instances of misconfigured automations that frequently failed.[4] As examples, the well-configured automations included jobs that ran: (i) every six minutes for a year, (ii) every ten minutes for two months, (iii) every half hour for two months.

**Deriving the model.** Given the cleaned data, Figure 6 then presents the distribution of GFI (using $n = 100$ logins) for five different years. We see a possible trend towards overall less failure from 2005–2008, but 2009 reverses this drift, so we do not attempt to model the prevalence of failure as a function of time.

The figure inset shows the empirical density for 2010 along with two synthetic datasets. First, we fitted a binomial distribution to the 2010 data and randomly generated a new dataset of equal size from that distribution. Second, we applied the same process but

[4] We found it interesting that in some of these cases, when contacted the local administrators were unaware or had forgotten about the existence of this automation.

instead used a beta-binomial distribution. We see from the inset that the actual data exhibits more variance than we can capture using a binomial model. The beta-binomial model provides a significantly better fit as it allows for an extra variance factor termed *over-dispersion*. Beta-binomial is the predictive distribution of a binomial random variable with a beta distribution prior on the success probability, i.e., $k \sim \text{Binomial}(p, n)$ where $p \sim \text{Beta}(\alpha, \beta)$. Then for a given $n$, $\alpha$ and $\beta$, we have:

$$k \sim \binom{n}{k} \frac{\text{Beta}(k+\alpha, n-k+\beta)}{\text{Beta}(\alpha, \beta)}$$

We can interpret the success of this fitting in terms of lack of independence. If all login attempts were IID, then we would expect to capture GFI effectively using a binomial distribution. The need to resort to a beta-binomial distribution indicates that the random variables lack independence or come from different distributions, such that the probability of success has a beta-prior instead of being constant. This makes sense intuitively because (i) different users will have different probabilities of success, and (ii) the login attempts from a single user are not independent: one failed login attempt affects the probability of success of the next login attempt (negatively if the user has forgotten their password, positively if they simply mistyped it).

# 6. EVALUATION

In this section we apply our detection procedure to the extensive LBNL dataset. We discuss parameterizing the detector, assess its accuracy, and characterize the attacks it finds, including whether the attacks appear targeted or indiscriminant.

## 6.1 Parameterization

Our procedure first requires selecting a mean shift $\Delta\mu$ that we wish to detect. We set this to 10 failed logins per event of 100 logins, basing our choice on the stealthiest attack we wish to detect without overburdening the analyst. On average this site sees 500 logins per day, so a threshold of $\Delta\mu = 10$ bounds the number of attempts a brute-forcer can on average make without detection to 45 (9 attempts $\times$ 5 events) spread over a day. Fitting our beta-binomial distribution (§ 5) to the 2005–2008 data yields the parameters $\mu = 7$ and $\sigma = 4.24$, and so our chosen value corresponds to a shift in mean of approximately $2\sigma$. (Note that this is *different* from stating that we detect a "two sigma" event, because due to the cumulative nature of the detection process, it takes significantly more perturbation to the site's activity than simple stochastic fluctuations to reach this level.)

We choose the other parameter, the decision threshold $H$, based on computing ARLs using the Markov chain analysis sketched in § 4.1. Table 3 shows the *in-control* and *out-of-control* ARLs for $k = 5$ and varying values of $H$. (We use $k = 5$ based on the rule-of-thumb of setting $k = \frac{\Delta\mu}{2}$ [10].) Given these results, we choose $H = 20$, as this gives a quite manageable expected false alarm rate of one-per-3,720 events, which, given that the site produces about 5 events per day, translates to an average of two alarms per year, and thus an expected 16 false alarms for the complete dataset. This level detects actual stealthy attacks after 5 events (50 brute-forcer login attempts, since the computation is for a shift in the mean of $\Delta\mu = 10$). In a practical setting, $H = 10$ (one false alarm per month) could work effectively.

To validate the assumptions underlying our detection model, we ran the CUSUM detector on the "cleaned" data (per § 5) to compare the expected false alarms with empirical false alarms. The detector flagged a total of 12 false alarms, reflecting cases where the failure of benign users lead to the alarm.

| H | In-control ARL | Out-of-control ARL |
|---|---|---|
| 1 | 9 | 1 |
| 10 | 144 | 3 |
| 20 | 3,720 | 5 |
| 30 | 99,548 | 7 |
| 40 | 2,643,440 | 9 |

Table 3: In-control and out-of-control ARLs for $k = 5$ and varying values of $H$.

## 6.2 Assessment of Detection

The two components of our detector can each exhibit false alarms: *false coordinated attack epochs* and *false attack participants*. We can readily identify the former by inspection, as incorrect attack epochs can manifest in one of three ways: (i) the epoch consists of a singleton brute-forcer and a collection of legitimate users who had failures, (ii) the epoch consists of non-coordinating brute-forcers having no apparent coordination glue, and (iii) bad luck: the epoch consists of just legitimate users who failed. The latter kind of false alarms (false attack participants) pose a harder challenge to classify, given we lack ground truth. Since LBNL didn't itself detect and assess the majority of the attacks we detect, we use the following heuristic to gauge whether our procedure correctly classified a remote host as a brute-forcer. We inspect the the host's login activity in the attack epoch along with its future activity. If none of this succeeded, we can with high confidence deem that host as a brute-forcer. For hosts that ultimately succeed, we confirm whether the success reflected a break-in by checking whether LBNL's incident database eventually noted the activity.

Running the procedure over 8 years of data, the *Aggregate Site Analyzer* detected a total of 99 attack epochs. After then processing these with the *Attack Participants Classifier*, we find nine[5] represent false alarms. We detect a total of 9,306 unique attack hosts participating in the distributed attack epochs, two of which succeed in breaking-in. The procedure classified only 37 benign hosts as attack hosts, reflecting a very low false alarm rate. On days that include an attack epoch, we find on average about 100 benign $\langle R, U \rangle$ pairs filtered out using our past-history assessment, but on average only about 1.7 "forgotten username" instances detected and removed.

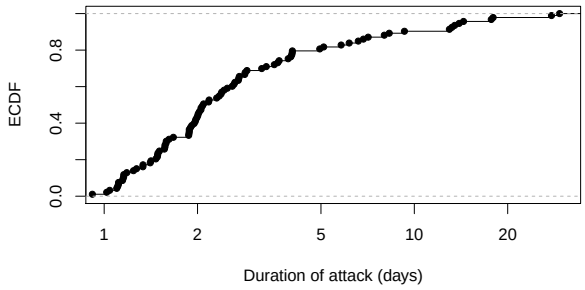


Figure 7: Empirical CDF of the duration of attacks (number of days)

[5]Note that this number differs from that found earlier on "cleaned" data because some of those false alarms coincided with actual attack epochs, and the *Attack Participants Classifier* then removed them due to a mismatch of coordination glue.

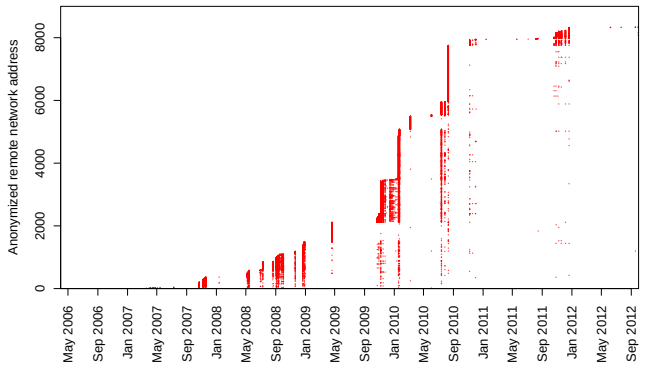


Figure 8: Participating attack hosts in the distributed attacks detected from 2005 to 2012 at LBNL.

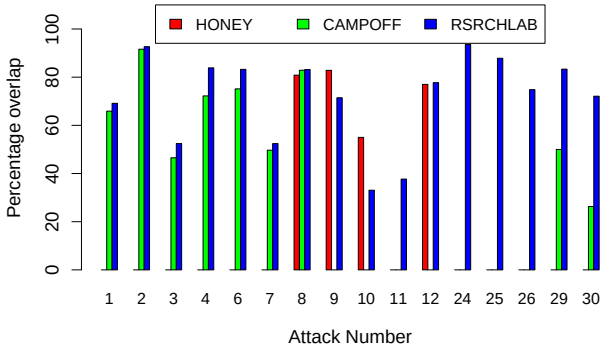


Figure 9: Percentage overlap of attack hosts seen at LBNL with that at sites HONEY, CAMPOFF and RSRCHLAB. The figure displays only the subset of attacks that appear in at least one of the three sites. (Note that none of the attacks appear in HOMEOFF).

Figure 7 shows the empirical CDF of the span of detected attack epochs. These coordinated attacks often span multiple days, and sometimes multiple weeks. The majority of the attacks exhibit strong coordination glue in terms of either the set of local machines probed or the usernames targeted for brute-forcing. Of the 90 true attack epochs, 62 have common-set-of-local-machines glue and 25 have username-"root" glue. Only 3 epochs did not manifest any glue we could identify; these epochs probed machines across a wide range of addresses and using a dictionary of generic usernames, such as *mysql* and *admin*.

Figure 8 shows the attack hosts participating in the various distributed attack epochs over time, where we number distinct hosts consecutively starting at 1 for the first one observed. The significant overlap of attack hosts across attack episodes shows that many of these attacks employ the same botnet. We then analyzed the coordination glue in these attack epochs to consolidate the set of epochs into *attack campaigns*. We use the following rules to group epochs into campaigns based on observing evidence that the same attacker conducting different attack epochs that work towards the same goal: (i) epochs with the same common-set-of-local-machines coordination glue, and (ii) epochs appearing on the same day with username-*root* coordination glue. Our detector considers these as multiple attack epochs rather than a single attack because this is indeed how the campaign proceeds, stopping for a few hours/days and then reappearing. Using these rules, we group the 62 attacks with common-set-of-local-machines glue into 12 distinct attack campaigns. Only a few of the 25 epochs group using heuristic (ii), condensing the root-set to 20 campaigns. This leaves us with a total of 35 attack campaigns.

Table 4 summarizes the magnitude, scope and stealthiness of the attacks we detect. All of these attacks were stealthy when observed from the viewpoint of individual hosts; on average the attack hosts made $\approx 2$ attempts per local machine per hour. We can however detect a large fraction of these attack campaigns using a point-wise network-based detector that looks for high-rate hourly activity in terms of either the total number of failed attempts or the number of local hosts contacted. Note that we also detect attacks that a site cannot detect using either host-based or network-based point-wise detection (campaigns 5, 7 and 8 in Table 4). Finally, two of the campaigns succeeded, the first of which (campaign 1) as best as we can tell went undetected by the site.

We also find a difference in the characteristics between attacks that have set-of-local-machines coordination glue versus the ones that only have username-*root* glue. The latter tend to target a wide range of the site's address space and often involve just a few attack hosts brute-forcing at a high rate. Attacks having set-of-local-machines coordination glue often exhibit the pattern of the attackers stopping and coming back. We did not find any sequential pattern in any of these campaigns; rather, the attackers targeted servers spread across the address space, often including addresses in both of LBNL's distinct address blocks. We also did not find any pattern among the local servers in terms of belonging to the same research group or compute cluster.

## 6.3 Establishing the scope of attacks

Next, we attempt to establish which attacks specifically targeted the LBNL site versus global attacks that indiscriminantly probed the site. To do so we look for whether the attack hosts of a given campaign appeared in any of our four correlation datasets, HONEY, RSRCHLAB, HOMEOFF, and CAMPOFF.

We find that 16 campaigns appear in at least one of these four datasets. These include five username-root coordination glue attacks and all but one of the attacks with set-of-local-machines coordination. Figure 9 plots the percentage overlap of the attack hosts detected in the global attacks at LBNL with that at other sites, showing a high overlap in most cases. We investigated campaign 5, which does not appear at any of the other sites, and found that it indeed targeted LBNL, as the attack hosts all probed a set of six usernames each valid at the site. As shown by the hourly rates in Table 4, this targeted attack also proceeded in a stealthy fashion, with each remote host on average making only 9 attempts and contacting 3 local servers per hour. It's possible that some of the other campaigns also specifically targeted LBNL, though for them we lack a "smoking gun" that betrays clear knowledge of the site.

Finally, to give a sense of the nature of global attacks, Figure 10 shows the timing patterns of login attempts at the LBNL and HONEY sites during part of campaign 8. From the clear correlation (though with a lag in time), we see that the activity at both reflects the same rate (which varies) and, for the most part, the same active and inactive periods.

# 7. CONCLUSION

In this work we propose a general approach for detecting distributed, potentially stealthy activity at a site. The foundation of the method lies in detecting change in a *site-wide* parameter that summarizes *aggregate* activity at the site. We explored this approach in concrete terms in the context of detecting stealthy distributed SSH brute-forcing activity, showing that the process of legitimate users

| ID | Appearances | Attrs. | Aggregate statistics | | Per remote avg. hourly characteristics | | |
|---|---|---|---|---|---|---|---|
| | | | Attack machines | Local machines | Attempts | Locals contacted | Per-Local attempts |
| 1 | 2007: [Jul 7-9], [Oct 20-23], [Nov 5-9](2), [Nov 13-18](2) | **L,!!** | 431 | 133 | 74.68 | 56.10 | 1.33 |
| 2 | 2008: [Apr 30 - May 7],[May 8-14](3) | **L** | 286 | 140 | 98.50 | 54.80 | 1.79 |
| 3 | 2008: [Jun 28-29], [Jun 30 - Jul 1] [Jul 7-9], [Aug 17-21], [Sep 1-8] (5) | **L** | 969 | 113 | 293.30 | 41.70 | 7.00 |
| 4 | 2008: [Sep 8-13](3) | **L** | 378 | 257 | 52.50 | 40.70 | 1.28 |
| 5 | 2008: [Sep 16-18] | **L,S,T** | 88 | 12 | 9.00 | 2.53 | 3.57 |
| 6 | 2008: [Sep 23-26](2), [Sep 29 - Oct 2](2) | **L** | 185 | 109 | 48.50 | 38.38 | 1.26 |
| 7 | 2008: [Nov 18-19], [Nov 20 - Dec 29](5) 2009: [Apr 7-9] | **L,S** | 1,097 | 22 | 16.01 | 8.04 | 1.99 |
| 8 | 2009: [Oct 22-23], [Oct 27 - Nov 24](5) | **L,S** | 1,734 | 5 | 5.60 | 3.70 | 1.50 |
| 9 | 2010: [Dec 6 - Jan 10](6), [Jan 11-18], [Jan 20-22], [Mar 4-8] | **L** | 3,496 | 44 | 38.80 | 21.50 | 1.80 |
| 10 | 2010: [Jun 16 - Jul 27](2), [Jul 29 - Aug 11] | **L** | 7,445 | 1,494 | 90.80 | 34.50 | 2.70 |
| 11 | 2010: [Nov 1-6] (2), [Nov 7-8], [Nov 27 - Dec 1], [Dec 15-17] | **L,!** | 581 | 98 | 140.60 | 45.47 | 3.09 |
| 12 | 2011: [Oct 11-19], [Oct 25-29](2), [Nov 4-7], [Nov 17-20] | **L** | 377 | 158 | 33.93 | 25.25 | 1.34 |
| 13 | 2010: [Mar 30 - Apr 1] | **R,t** | 78 | 18,815 | 999.70 | 118.91 | 1.33 |
| 14 | 2010: [Apr 23-26] | **R,t** | 130 | 29,924 | 2325.57 | 117.97 | 1.22 |
| 15 | 2010: [May 7-10] | **R,t** | 72 | 9,300 | 713.05 | 67.47 | 1.36 |
| 16 | 2010: [Sep 20-22] | **R,t** | 33 | 5,380 | 69.05 | 60.72 | 1.14 |
| 17 | 2010: [Dec 27-30] | **R,t** | 32 | 3,881 | 260.59 | 43.11 | 1.34 |
| 18 | 2011: [Feb 10-14](2) | **R,t** | 108 | 7,520 | 40.45 | 27.21 | 1.48 |
| 19 | 2011: [May 16-18] | **R,t** | 30 | 1,621 | 153.23 | 19.70 | 2.02 |
| 20 | 2011: [Jul 21-22] | **R,t** | 20 | 2,556 | 388.25 | 38.13 | 1.18 |
| 21 | 2011: [Aug 2-6] | **R,t** | 45 | 9,465 | 315.12 | 21.66 | 2.41 |
| 22 | 2011: [Aug 7-9] | **R,t** | 48 | 6,516 | 444.16 | 17.60 | 2.18 |
| 23 | 2011: [Aug 17-21](2) | **R,t** | 22 | 3,279 | 33.07 | 16.40 | 2.02 |
| 24 | 2011: [Nov 2-4] | **R** | 31 | 3,446 | 273.80 | 20.08 | 1.02 |
| 25 | 2011: [Nov 30 - Dec 5] | **R** | 181 | 10,467 | 829.68 | 18.31 | 1.03 |
| 26 | 2011: [Dec 18-20] | **R** | 258 | 961 | 1099.85 | 14.00 | 1.02 |
| 27 | 2012: [Jul 20-21] | **R,t** | 2 | 53,219 | 20,844 | 11,749 | 1.06 |
| 28 | 2012: [Aug 27 - Sep 2] | **R,t** | 10 | 1,912 | 20.84 | 14.38 | 1.23 |
| 29 | 2012: [Sep 26-29] | **R** | 6 | 1,971 | 72.30 | 13.05 | 1.59 |
| 30 | 2012: [Oct 8 - Nov 1](4) | **R,S** | 190 | 19,639 | 5.27 | 4.97 | 1.06 |
| 31 | 2012: [Nov 16-18] | **R,t** | 3 | 493 | 38.36 | 12.22 | 2.99 |
| 32 | 2012: [Nov 30 - Dec 2] | **R,t** | 3 | 344 | 133.00 | 68.80 | 1.93 |
| 33 | 2008: [Jan 9-12] | **X,t** | 17 | 63,015 | 2,846.44 | 1,761.69 | 1.61 |
| 34 | 2011: [Apr 8-26] | **X,t** | 67 | 19,158 | 591.34 | 87.41 | 6.76 |
| 35 | 2012: [Dec 14-17] | **X,t** | 13 | 45,738 | 1,490.26 | 1,430.67 | 1.04 |

Table 4: Characteristics of the detected coordinated *attack campaigns*. In *Appearances*, numbers in parentheses reflect how many attack epochs occurred during the given interval. *Attrs.* summarizes different attributes of the activity: **L** = coordination glue was set of local machines, **R** = coordination glue was username "root", **X** = no discernible coordination glue, **S** = stealthy, **T** = targeted, **t** = possibly targeted but no corroborating evidence, **!** = successful, **!!** = successful and apparently undetected by the site.

failing to authenticate is well-described using a beta-binomial distribution. This model enables us tune the detector to trade off an expected level of false positives versus time-to-detection.

Using the detector we studied the prevalence of distributed brute-forcing, which we find occurs fairly often: for eight years of data collected at a US National Lab, we identify 35 attack *campaigns* in which the participating attack hosts would have evaded detection by a pointwise host detector. Many of these campaigns targeted a wide range of machines and could possibly have been detected using a detector with a site-wide view, but we also find instances of stealthy attacks that would have proven very difficult to detect other than in aggregate. We correlated attacks found at the site with data from other sites and found many of them appear at multiple sites simultaneously, indicating indiscriminant global probing. However, we also find a number of attacks that lack such global corroboration, at least one of which clearly targeted only the local site. Some campaigns in addition have extensive persistence, lasting multiple months. Finally, we also find that such detection can have significant positive benefits: users indeed sometimes choose weak passwords, enabling brute-forcers to occasionally succeed.

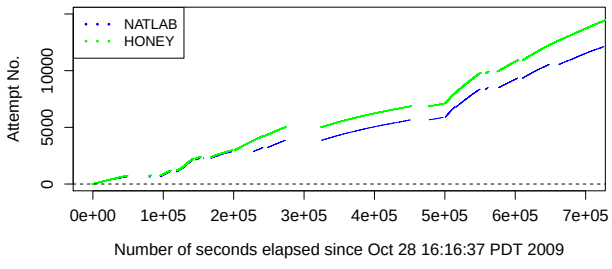


Figure 10: Timing of login attempts at HONEY machine and LBNL sites during part of attack number 8 (Oct 2009 - Nov 2009). The plot is based on data for only one of the machines targeted during the attack at LBNL.

## Acknowledgments

Our thanks to Mark Allman, Peter Hansteen, and Robin Sommer for facilitating access to the different datasets required for this work. Our special thanks to Aashish Sharma for running down

various puzzles and to Partha Bannerjee and James Welcher for providing crucial support for the processing of the LBNL dataset.

This work was supported by the U.S. Army Research Office under MURI grant W911NF-09-1-0553, and by the National Science Foundation under grants 0831535, 1161799, and 1237265. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

# 8. REFERENCES

[1] BlockHosts. http://www.aczoom.com/blockhosts/.

[2] DenyHosts. http://denyhosts.sourceforge.net/.

[3] sshguard. http://www.sshguard.net/.

[4] The Hail Mary Cloud Data - Data collected by Peter N. M. Hansteen (peter@bsdly.net). http://www.bsdly.net/~peter/hailmary/.

[5] ICS-ALERT-12-034-01 — SSH Scanning Activity Targets Control Systems. http://www.us-cert.gov/control_systems/pdf/ICS-ALERT-12-034-01.pdf, Feburary, 2012.

[6] R. Bezut and V. Bernet-Rollande. Study of Dictionary Attacks on SSH. Technical report, University of Technology of Compiegne, http://files.xdec.net/TX_EN_Bezut_Bernet-Rollande_BruteForce_SSH.pdf, 2010.

[7] D. Brook and D. A. Evans. An approach to the probability distribution of CUSUM run length. In *Biometrika*, volume 59, pages 539–549, 1972.

[8] C. Gates. Coordinated scan detection. In *16th Annual Network and Distributed System Security Symposium*, 2009.

[9] D. Gerzo. BruteForceBlocker. http://danger.rulez.sk/projects/bruteforceblocker.

[10] D. M. Hawkins and D. H. Olwell. Cumulative sum charts and charting for quality improvement. Springer, 1998.

[11] L. Hellemons. Flow-based Detection of SSH Intrusion Attempts. In *16th Twente Student Conference on IT*. University of Twente, January 2012.

[12] C. Jacquier. Fail2Ban. http://www.fail2ban.org.

[13] M. Kumagai, Y. Musashi, D. Arturo, L. Romana, K. Takemori, S. Kubota, and K. Sugitani. SSH Dictionary Attack and DNS Reverse Resolution Traffic in Campus Network. In *3rd International Conference on Intelligent Networks and Intelligent Systems*, pages 645–648, 2010.

[14] E. L. Malecot, Y. Hori, K. Sakurai, J. Ryou, and H. Lee. (Visually) Tracking Distributed SSH BruteForce Attacks? In *3rd International Joint Workshop on Information Security and Its Applications*, pages 1–8, Feburary, 2008.

[15] J. Owens and J. Matthews. A Study of Passwords and Methods Used in Brute-Force SSH Attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[16] A. V. Siris and F. Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. In *IEEE GLOBECOM*, pages 2050–2054. IEEE, 2004.

[17] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. In *7th ACM Conference on Computer and Communications Security*, Athens, Greece, 2000.

[18] J. Vykopal, T. Plesnik, and P. Minarik. Network-based Dictionary Attack Detection. In *International Conference on Future Networks*, 2009.

[19] H. Wang, D. Zhang, and S. K. Detecting SYN flooding attacks. In *21st Joint Conference IEEE Computer and Communication Societies (IEEE INFOCOM)*, pages 1530–1539, 2002.

[20] C. M. Zhang and V. Paxson. Detecting and Analyzing Automated Activity on Twitter. In *Passive and Active Measurement*. Springer, 2011.