DETECTING UNAUTHORIZED USE IN ONLINE JOURNAL ARCHIVES: A CASE STUDY•

Paul Seligman
Department of Computer Science
Dartmouth College, Hanover NH 03755 USA
Paul.Seligman@alum.dartmouth.org

Sean Smith
Department of Computer Science
Dartmouth College, Hanover NH 03755 USA
sws@cs.dartmouth.edu

ABSTRACT

JSTOR is a not-for-profit online library containing a full back-run of digitized versions of a large number of academic journals. In order to help defray costs for maintaining the archive, subscribing institutions (such as libraries and universities) pay a fee to enable their users to access it. However, in order to make this access easy for authorized users—and to avoid requiring changes to the current IT infrastructure of their subscribing institutions—JSTOR authenticates users via the IP address of the computer that generated the request. (If the IP address belongs to a subscribing institution, the user is granted access.)

This design decision introduces the potential for trouble: unauthorized users can access the archive if they can find an unprotected proxy machine at a subscribing institution and request material via that machine. (Observant archive staff have noticed abnormal usage patterns and traced them to such unauthorized use.) Unfortunately, this design decision also constrains potential countermeasures: since we cannot change the infrastructures of the subscribers, we instead need to have the archive itself try to detect and respond to incidents of fraudulent use.

In this paper, we describe our experiments to automate these countermeasures.

KEYWORDS

Machine learning, fraud detection, digital libraries.

1. INTRODUCTION

1.1 Digital Libraries

In the standard model, scholars write their results in papers, which—after a careful process of peer-review and editing—are published in archival journals, for subsequent scholars to browse and read. In the traditional instantiation, these journals exist as paper in a bricks-and-mortar library. An institution such as a university builds such a library for its community of scholars, and subsidizes the journal process through subscription fees.

The physical nature of the library and its contents can be expensive and limiting. However, the physical nature can also limit potential malfeasance: the institution can limit access to the building to the appropriate scholarly community (which, in some cases, might be anyone who walks in the door) and screen for users who remove material without authorization. The high marginal cost of duplicating paper journals can discourage other forms of large-scale theft.

In the electronic age, these limitations of bricks-and-mortar and paper no longer need apply. An electronic setting frees an institution from constructing a large building to hold fragile paper journals, and enables scholars to use digital search tools and to access the library from anywhere on campus—or even from a

• This research has been supported in part by the Mellon Foundation, NSF (CCR-0209144), AT&T/Internet2 and the Office for Domestic Preparedness, Department of Homeland Security (2000-DT-CX-K001). This paper does not necessarily reflect the views of the sponsors. Although the authors are grateful for the discussion and access logs that JSTOR provided for this project, the material in the paper should not be construed as an official statement expressing the views or practices of JSTOR.

remote site. This freedom creates a role for a third-party archive that provides content to subscribing institutions. In turn, these institutions provide access to their scholars per local policy, through their local information infrastructure. However, an electronic archive also removes the inherent protections of physical media: the marginal cost for making a perfect copy is zero, and an institutional library no longer has a clearly defined front door to screen which scholars come in and what they remove.

1.2 JSTOR

This paper examines some security aspects for one such third-party archive, JSTOR (www.jstor.org; see [2, 3, 7] for overviews). JSTOR is a non-profit digital library that contains over 350 journal titles and provides this content to over 1800 subscribing institutions. Scholars at these subscribing institutions can use a Web front-end to browse, search, and download articles from the JSTOR archive. In order to increase the usability of this archive by scholars, and to let institutions retain flexibility in how they manage their local information infrastructure, JSTOR delegates access control to the institution: if the institution lets the scholar into its local network, then JSTOR will recognize that scholar as legitimate. (This approach also frees users from having to enroll in and remember a JSTOR-specific form of authentication, and frees an institution from having to modify their systems in order to provide access to JSTOR.)

To implement this design, JSTOR determines whether a request is valid based on the IP address of the originating machine. Within the Internet, each individual institution owns some range of the IP address space. Machines at that institution are identified by addresses within that range. At first glance, one might assume that since only one user uses a machine such as a laptop or desktop at one time, requests that come from a specific address will represent the action of one individual scholar sitting in front of the machine. (Things like dynamically assigned IP addresses and public-access machines may cause this matching to change over time.) However, the subscribing institutions offer two significant departures from this model. Some institutions funnel traffic through a gateway machine (thus the IP address of this gateway will aggregate traffic from many users). To accommodate traveling users, some institutions set up proxy machines. The traveler connects to the proxy, which then issues the request. Figure 1 sketches this basic architecture.

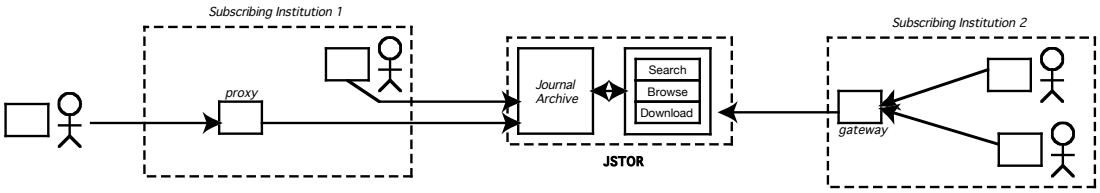

Figure 1: A sketch of the JSTOR architecture: users access the archive through machines in the IP address spaces of their institutions, which is how requests are authorized. Typically, one user uses one machine at a time, so the traffic one IP address represents one user at that site. However, some institutions funnel traffic through *gateways*. Some set up *proxies* so scholars can still access the archive (and other information services) when on travel.

1.3 Problem

This approach to authorization is based on the assumption that if a request came from a machine owned by institution *X*, then the user who issued that request is a legitimate scholar at *X* (since *X* let him or her onto its local network), and therefore the request is valid. However, the existence of proxies that are *open* (have no authentication) or have weak authentication threatens this assumption. If a user not affiliated with a subscribing institution discovers such a weak proxy, he or she can use it to issue fraudulent requests to the archive—which would regard the requests as authorized, since they originated at an authorized address. Furthermore, if such a user were particularly malicious, he or she could use the proxy as an avenue to steal parts of the archive, by systematically downloading it.

This is the main threat model: not occasional users making occasional illegitimate accesses, but rather serious adversaries systematically trying to copy a large portion of the content. Observant archive staff have noticed such usage patterns, and traced them to illegitimate remote access to open proxies at subscribing institutions. Given the relatively penetrable state of the current computing base, we must consider any machine at the institution—not just proxies—as a potential conduit of such attacks.

1.4 Solution Constraints

We were handed the logs, and an oral description of fraudulent behavior (e.g., "pirates are trying to steal our archives").

The natural security solution would be to find and shut down all open proxies, and to require stronger authentication of all users. However, these approaches do not fit within the design constraints of making the archive easy to use by scholars, and easy to adopt by subscribing institutions. Changing how the subscribing institutions manage their IT infrastructure is not allowed.

Given these constraints, we need to look at the server side for a solution. Depending on human staff to manually wade through the access logs and identify fraud is not feasible, and does not permit identification in time for an effective response. What technologies can we utilize to automate defense? In this setting, false positives are acceptable, since a "fraud" label merely marks a set of behavior for subsequent human analysis.

In research arenas, the areas of *machine learning (ML)* and *artificial intelligence (AI)* have yielded computerized techniques to grapple with identifying patterns in large, complex data sets. As a consequence, we began a series of experiments to see if these techniques would help automate fraud detection in the JSTOR digital library. This paper reports our work. Section 2 examines the space of building blocks in ML and AI we considered. Section 3 describes how we transformed our problem and data into a form amenable for these tools. Section 4 describes our experiment plan, and what happened when ran these experiments on access logs from a sample month. Section 5 concludes with some avenues for future work, and implementation considerations.

2. BUILDING BLOCKS

2.1 General Framework

Suppose we have a large data set X and a function $F : X \rightarrow \{0,1\}$, and that we would like to write a computer program P that carries out the action F, at least approximately. (That is, $P(x) = F(x)$ with high probability, for $x \in X$.) For many real-world problems, it might not be clear how to write such a program P (e.g., suppose X was a set of photographs, and $F(x) = 1$ when x contained a tank). It might not even be clear how to precisely specify F, let alone develop a program P (e.g., suppose X was the set of income tax returns, and $F(x) = 1$ when x was fraudulent).

In the ML approach, we instead develop a parameterized family of programs P_y, and a set of techniques to determine whether a given P_y is "close" to F, and, if not, how to adjust to a z such that P_z is closer. Thus in some sense, the program "learns" how to do F by starting at some initial P_y, and systematically tuning itself based on past performance until it gets close to F.

In practice, ML offers a wide range of techniques that vary in how they construct the programs, how they carry out the training, and how much human involvement is involved. Some require a priori human analysis: extracting the right *feature set* of the instances of X to feed to the program. Human experts can also give some hints or descriptions of when $F(x)$ should take on the various output values; or perhaps in providing an explicit *labeled training set*: a sizable subset $X' \subset X$, with each $x \in X'$ labeled with $F(x)$. With some techniques, after the computer evolves to an effective P_z, we can transform the final z into some insightful characterization of the problem space; with others, the final z is meaningful only to the computer.

Because of its frequent effectiveness at finding needles (e.g., fraudulent action) in the haystacks of complex data sets, the use of machine learning has a long history in fraud detection (e.g., [5, 10] give two snapshots). Can we use it to help with this new fraud problem in emerging digital libraries? We first survey some of the main techniques. ([9] provides a good overall reference.)

2.2 Decision Trees

The *decision tree* technique requires a training set with predetermined fields and labels: that is, human experts have already figured a way to encode each instance of X, and have already calculated F on a sufficiently large and representative subset. When learning, our program chooses the field most likely to split

the training population in two, and creates a *decision node* to represent that split. The decision is applied to the entire population at that node, and that population is split on the result of that decision. This process of choosing a decision field, and splitting the population based on that field, recurses on each newly formed child node, over the data corresponding to that particular node. This recursive creation of nodes creates a tree. A branch of the decision tree is considered exhausted when all the inputs at the leaf have the same classification, or until all the possible fields have been exhausted.

2.3 Artificial Neural Networks

Artificial Neural Networks also start with a pre-determined method of encoding problem instances as fields, and a labeled training set. (Jain et al [8] give an extremely thorough overview.) Each input field is treated as an input node with a numeric value. These nodes each have an output, which is some numerical combination of their inputs. These first-layer nodes are then fed into a second layer of nodes, with each node in the previous layer contributing its output as an input to each node in the current layer. This can continue for as many internal node layers as found useful, with each layer having its own optimal number of nodes. This network of nodes constitutes the program P; the weights of the output-to-input connections between the nodes of different layers constitute the parameter y. $P_y(x)$ is the value of the last node, when x is encoded as a tuple of field values and fed to the first layer. The learning mechanism works by adjusting the inter-node weights until the output of the network is found to be the label of the input instance, for a sufficiently large part of the training set. However, unlike decision trees, the final weightings used in an accurate network do not tell a human analyst much about the underlying principles of the problem.

2.4 Clustering

Unlike the above two methods, *clustering* does not require the training set to be labeled, although it does require a pre-determined way to encode instances into fields. The idea behind clustering is not to predict the outcome of a particular learning instance, but rather to use the whole of the learning set to separate out instances into two or more distinct groups. We can then apply this separation technique to new data. The classic model of clustering is the *k-means* clustering model. In this model, the parameter y is a set of k points in the input vector space that are the centers of each cluster. P_y assigns an instance to the cluster whose center is closest. In the learning process, we start with an initial set of cluster centers, and sort the training set according these. Then, for each resulting cluster, we calculate the *centroid* (mean) of the set of instances assigned to this cluster. We use the set of cluster centroids as the cluster centers, and repeat the process. The repetition stops when the same instances are assigned to the same clusters in consecutive rounds. The technique then assumes that the cluster centers have stabilized, and will remain constant thereafter. A center of a cluster is, in some sense, the "archetype" of that set, in that encoding. This relation gives more avenues for human involvement. Boldon et al [1] give an example of using clustering for fraud detection in credit cards.

2.5 Genetic Algorithms

A *genetic algorithm* is an artificial intelligence technique inspired by nature. (Forrest and Mitchell [6] have a nice overview.) In a genetic algorithm, potential solution programs P are structured as sets of modular genes; the parameter y is the particular set for P_y. As before, we require a pre-determined way to encode problem instances as tuples of field values, to feed to these programs. We also require a way to quantitatively measure the *fitness* of a particular P_y – for example, by measuring how closely it approximates F on some labeled training set. The learning process proceeds over many *generations*. Each generation begins with a large population of potential P_y. We evaluate the fitness of each solution in this population, and mimic natural selection by propagating only the more fit solutions into the next generation. As part of this propagation, we also mimic sex by *crossover* (randomly swapping parts from one solution encoding to another). We also randomly *mutate* the solutions, keeping each within the parameters of the encoding scheme. The process continues from generation to generation until no significant gains are found from one generation to the next. The ability of genetic algorithms to converge quickly from random initial beginnings to a meaningful

conclusion depends on the encoding of that problem and solution space. The ability also depends on the solution encoding being large enough so that, initially, a mutation can significantly affect a solution's fitness.

3. FOUNDATIONS

How can we put the pieces of Section 2 together to solve the problem?

In the JSTOR case, we have massive amounts of data: the requests generated from each allowed IP address. We want to compute the function *F* that indicates whether the traffic from a given IP address is "normal" (i.e., probably from a legitimate user) or "abnormal" (perhaps fraudulent, and meriting further staff scrutiny). Although we have plenty of training data (the access logs), none of is labeled.

This is unfortunate, but is a fact of life: we were handed the logs, and an oral description of fraudulent behavior (e.g., "pirates are trying to steal our archives").

Without labeled data—but with the archive's opinions about what might constitute unauthorized use—the natural choice is clustering. Since we have plenty of human insight into what might constitute abnormal behavior, we can embody that as the initial cluster centers. However, since the data we have is fairly complex—access logs—it would behoove us to transform it into a set of fields more conducive to this classification. Hence, we begin with an analysis of the structure of the data, the field set we developed, and where our initial "normal" and "abnormal" archetypes may lie in this space.

3.1 Starting Point

Our training (and experiment) set was a previously recorded set of logs, representing all requests to archive servers over the month of September 2002. The information used from each archive log entry was the client IP address, and the action requested: download, search, or other. If the request was for a download, the program also noted the ISSN, issue, and article numbers of the document being downloaded. Our program worked by first collecting all pertinent data from the requests, and sorting by client IP address. This resulted in an array of structures, each representing the whole of one client's actions. We then analyzed each structure, to see how we could transform them into sets of fields to be used for the clustering algorithm. Through manual analysis, we sketched what an IP address with typical "normal" usage might look like, and what one with typical "abnormal" usage might look like. We then identified a series of metrics in which these scenarios would differ significantly.

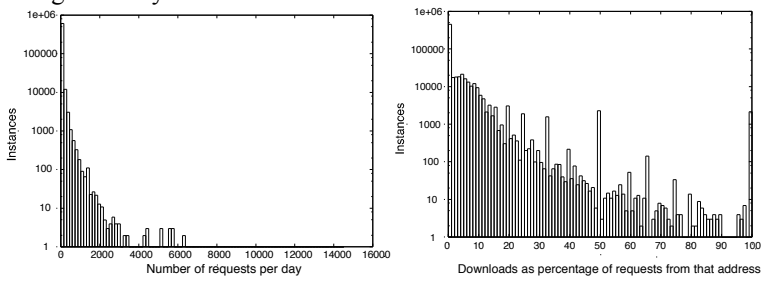

Figure 2: On the left, the distribution of number of daily requests from an IP address; on the right, the distribution of downloads, as a percentages of daily requests from an address.

3.2 Amount of Usage

A key division between normal and abnormal usage was the amount of usage from an IP address in a given day. The abnormal archetype we expected—an adversary subverting the archive's security methods to gain access to large amounts of data quickly—would involve heavy downloads of sequential articles. In Figure 2 (left), which shows the distribution of requests per day, we can see a significant number of outliers from the general downward sloping curve. Though not visible due to the graph's range, we found one IP address in our training records that had made over 14,500 requests in one day. Unfortunately, this attribute by itself cannot determine abnormal usage patterns. For example, as we noted earlier, many schools use gateways to

channel their outgoing web traffic. As a result, some IP addresses that contributed heavily to traffic flow were found to represent the usage of a large number of people, and thus their apparent abnormality was misleading. For our initial archetypes, we decided that the "normal" IP address would have 5 downloads per day, and the "abnormal" address would have 300.

3.3 Download Percent

Next, we decided to look at how often a user downloaded a document from the server, compared to making other types of requests. As noted earlier, archive requests are primarily to search for a document, to view a document, or to download a document. Again, if the adversary was quickly downloading sequential material, the traffic from his or her IP address would tend toward a high percentage of downloads. In Figure 2 (right), which shows the distribution of downloads as a percent of the daily requests from an address, we can see that there are again some notable outliers from the general downward slope. Again, this too cannot be used on its own to determine abnormality. Users who submit one request, a download, are not to be considered abnormal, but would be by this attribute alone. For our initial archetypes, we decided that the "normal" address would have 10% downloads, and the "abnormal" address would have 75%.

3.4 Searches Percent

A natural partner to the download percentage field is the percent of searches, within the overall request traffic from an address. Large numbers of downloads associated with large numbers of searches was a common occurrence among users. Large numbers of downloads with relatively few searches was found to be a characteristic of abnormality. In Figure 3 (left), which shows the distribution of the search percentage, we can see the same downward sloping graph. Like the others, this criteria also cannot determine abnormality on its own: a user who makes no searches, perhaps by using the archive's browsing mechanism instead to find relevant articles, would appear to be abnormal using this criteria alone. For our initial archetypes, we decided that the "normal" address would have 40% searches, and the "abnormal" address would have 5%.

3.5 Range Downloaded

Another striking difference among users' downloads were the types of documents downloaded. The vast majority of users were found to download from a wide range of articles, spanning different journals and different issues within each journal. Other users were found to have downloaded sequential articles over sequential issues of the same journal. This latter behavior is in keeping in the spirit of our idea of the typical abnormal user, who downloads information sequentially.

To measure this range for each address, we computed the distance between the average download and the most common ISSN downloaded, when arranged in a sorted frequency table. More specifically, for a given IP address, we placed its downloads into a frequency table, in which each entry consisted of both an ISSN number corresponding to an article downloaded by that user, and the number of downloads of articles the user downloaded with said ISSN number. We then sorted the frequency table entries from the most heavily downloaded ISSN numbers to the least heavily downloaded. This naturally led to a series of table entries in which the number of downloads of each ISSN value were decreasing. We then created the notion of "average download range" as follows: each entry in our sorted table was assigned an increment of a zero-based index. Then, each individual download was considered to be an instance of that index. For example, if an address downloaded 5 articles from journal A, 3 from B, and 1 from C, it would have a average download range of $((0 * 5) + (1 * 3) + (2 * 1))/(5 + 3 + 1) = .55$. This number was a fair indication of the spread of journals over which a particular user downloaded. Figure 3 (right) shows the distribution of this average download range.

This metric too cannot be taken on its own to determine abnormality. A user who downloads one article demonstrates the minimum possible document download range, but would probably not be counted as an abnormal user. For our initial archetypes, we decided that an abnormal address would have a download range of less than 1.0 (as a lower average would represent a more systematic download). For example, an IP address that had downloaded evenly over articles with three different ISSN values would just be considered normal, as the average distance in that case would be exactly 1.0.

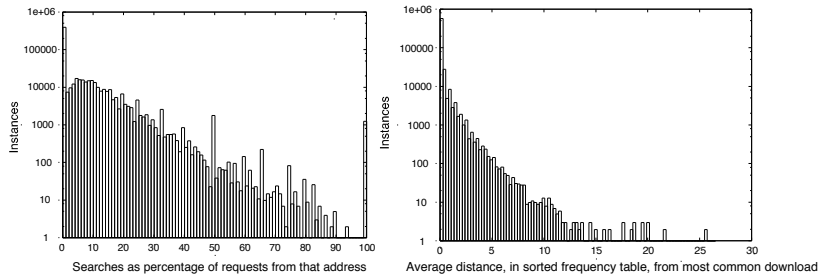

Figure 3: On the left, distribution of searches, as a percentage of daily requests from an address; on the right, distribution of download ranges: the distance between the average download and the most common ISSN downloaded, when arranged in a sorted frequency table

4. EXPERIMENTS AND RESULTS

4.1 Clustering

In Section 3, we developed a means to encode the activity associated with each IP address that might help an ML program distinguish legitimate access from fraudulent access, and we expressed characteristic archetypes of each, in terms of these fields. Our first experiment then is to use *clustering* on our training data, with these archetypes as the initial cluster centers.

In general, this approach was very adept at finding IP addresses in which abnormal usage was mixed with normal usage. Surprisingly, the program was just as adept at finding these mixed cases as it was at finding obviously abnormal usage, the type of which would be more easily noticed by humans.

The running time of this algorithm is $O(n^2)$, where n is the number of downloads over a given day. This is sufficiently fast to allow for real-time analysis of the server logs. In addition, this is fast enough to analyze the full logs of September, 2002 in under five hours, when run on an 800 Mhz PIII, with 256 MB of RAM, and running RedHat Linux 9. After writing the static clustering portion of this program, we passed a training set consisting of the given logs through the program as a progress benchmark. We found that 0.017% of IP addresses were classified as demonstrating abnormal usage. These IP addresses, though few in number, accounted for a striking 6.581% of articles downloaded over that same period. This shows that the set of abnormal IP addresses represented a large potential loss of throughput and data for the archive.

4.2 Dynamic Inputs

We were analyzing the daily traffic generated by an IP address. In the straightforward clustering approach, we examined this data after the traffic had completed. However, in real-world fraud detection, accepting data in a dynamic manner would be more useful, since it would permit classification of abnormal traffic in real time, while the traffic was occurring (and perhaps before significant amounts of material had been fraudulently downloaded.) Hence, the next step we wanted to try was to adapt our clustering approach to accept the IP address traffic incrementally. However, trying this technique raised new questions. How accurate will dynamic classification be? Will we place an address in one category, only to change its classification as more data arrives? How much of an address's traffic do we need to see before we can classify it?

We added dynamic input to our clustering program as follows: after every request from an IP address, that IP address was re-evaluated using our existing cluster centers. We then ran the same training set to gauge how quickly the program could correctly classify an IP address' behavior. It was found that less than 0.001% of IP addresses were first defined as abnormal, only to be reconsidered as normal by the end of the day's run. This is a very useful aspect of this program, as it shows that dynamically classifying users in this fashion will not lead to a large number of false positive cases that must be followed up on, only to have the users be considered normal again after more usage. On average, abnormal usage was detected after 65% of a user's articles were downloaded.

4.3 Genetic Algorithms

When we adjusted clustering for dynamic input of address traffic, the issues of speed and accuracy become important (as we have discussed). To a large extent, the performance of clustering (once we've decided the fields to use) depends on the archetypes we use as the initial cluster centers. If we started with better archetypes, we might be able to improve speed and accuracy for the dynamic version. How can we get better archetypes?

For this problem, we decided to try genetic algorithms. The genes were an archetype pair, for abnormal and normal clusters. We ran each gene against address traffic, fed dynamically, and measured its performance by looking at two values: what classification the gene suggested for this address, and how much of the address' traffic was required before it made this classification. We evaluated the overall fitness of a gene based on how well it would help the performance of our dynamic clustering. We started by running the clustering on a set of full traffic runs, to produce a labeled set. We then used this labeling to guide our fitness evaluation for a gene. We awarded points for the relative speed at which the gene correctly identified an abnormal address. We deducted points for falsely identifying a normal user as abnormal. We heavily deducted points for failing to identify an abnormal user. To discourage propagation of genes that would classify every IP address as abnormal, we required a minimum number of addresses that must be classified normal.

This algorithm was started with two genes that were hardcoded into the program. On each generation, it would run the given log file, and determine the fitness of each of the current genes. The best gene was passed into the next round. The remaining genes were selected pseudo-randomly, using a process that resulted in more fit genes being more likely to be chosen. Every pair of chosen genes was crossed at a random point in their structure. Finally, every value of every gene was, with a 12.5 percent chance, mutated. The mutation was implemented by adding or subtracting any amount between zero and twenty percent of the gene's current value at that point. If the gene's value at that point was already zero, a small constant was added to it. The number of genes was incremented every generation, until a predefined limit was reached. The number of generations over which to run was also a predefined input to the algorithm.

The output of a run of the genetic algorithm was a pair of improved normal/abnormal archetypes, which could be used for dynamic clustering on traffic unfolding in real time. (Fawcett et al [4] give an example of this same general approach—using ML to analyze full user logs, in order to develop better ways to predict fraud based on partial user logs—for the problem of telephone call fraud.) Due to the extended amount of time required for a genetic algorithm to asymptotically reach its optimal solution, only the first three days of logs were used as inputs to determine the optimal usage patterns. The genetic algorithm gravitated toward an abnormal usage archetypes that included 118 downloads per day, 93% of requests being downloads, and 0% of requests being searches. The normal usage pattern was found to have 1 download per day, 6% of requests being downloads, and 54% being searches. Running the same three days of logs with the archetypes chosen by the genetic algorithm found abnormal usage patterns after only 40% of the articles were downloaded, a marked improvement over the predefined archetypes chosen at the beginning of this program. This was a large improvement over the original archetypes, as it leads to less throughput and data loss before appropriate measures can be taken.

4.4 Distributed Attacks

Our manual analysis of the access logs showed a new type of abnormal behavior: *distributed components*. Apparently, the same adversary had accessed two different open proxies, and then partitioned a classic "download massive amounts of consecutive articles" attack between these two proxies.

In general, such a distributed attack will lessen the abnormal characteristics of any one IP address. By randomizing which machine downloaded which documents, and thereby decreasing the number of daily requests from these machines, usage of this sort becomes more difficult to recognize.

The most straightforward way to check for such an attack was to superimpose every pair of slightly abnormal user profiles. By assuming separate IP addresses represented a single user, we could then determine how abnormal such a superimposed, fictional user's usage would appear. If the superposition of the two profiles demonstrated a more abnormal usage pattern, with the average download range being the defining characteristic of increasing abnormality, our program can recognize this as an attempt at a distributed attack.

If a particular address exhibited significant abnormal behavior, then a naive attempt at superposition would indicate that it was conspiring with each of the other addresses—since their joint behavior would also be abnormal. To avoid this, we adjust the profiles of each IP address to take into account their relative number of downloads before superimposing them into the fictional user.

The program was extended to look for distributed component abnormal usage. Using the implementation described above, it was found that two percent of abnormal users were likely to be involved in a distributed components attack. This relatively large number indicates that such an addition to the program will be useful in a real world application.

5. CONCLUSIONS AND FUTURE WORK

Deploying a real-world digital library requires accommodating real-world usability and flexibility constraints—which, in this archive's case, led to the use of IP addresses for authorization. This approach leaves the archive open to attacks from trusted IP addresses that have been compromised. This form of attack has been successfully used to download articles without permission.

In our work, we explored the use of machine learning techniques to automate the detection of such misuse. This work showed that the use of a *k*-means clustering algorithm modified to include an understanding of the desired division dynamics can be effective in recognizing unauthorized library access—even recognizing this behavior while it is in progress. Using genetic algorithms (e.g., perhaps running overnight on the previous day's data) can produce archetype pairs that further improve this performance. Future work could include obtaining and using labels on an input set, in order to use additional ML algorithms, as well as exploring the space of automatic responses, once potential fraud is detected.

Perhaps the most attractive result of our work was the humble requirements of our program. For instance, if an archive were to implement this solution, it would require no more than one mid-range, dedicated machine. This machine could be used to parse server logs in real time, and respond as necessary. As abnormal usage patterns are found, this machine could instruct the servers to slow or stop all traffic to the questionable IP address, until such time as a human could verify the algorithm's findings. With genetic algorithms constantly updating the program's archetypes, the reliability of the program would increase over time.

For other institutions to adapt this model, more work would need to be done. Our means of turning Apache logs into meaningful inputs (e.g. where this request originated, and what type of request is it) would have to be adapted to each institution's setup and recording practices. After that, the program would need to be fed training sets, so that it could adapt to the particulars of an institution's traffic. After the initial cost of engineering time, however, this solution becomes economically attractive. Little hardware is needed, and hardware costs scale well with an institution's traffic.

REFERENCES

[1] R.J. Boldon and D.J. Hand, 2000. Unsupervised profiling methods for fraud detection. *Credit Scoring and Credit Control,* VII, September.

[2] W. Bowen, 2001. The Academic Library in a Digitized, Commercialized Age: Lessons from JSTOR. *ALA Midwinter Participants' Meeting*, January.

[3] R. Chepesiuk, 2000. JSTOR and Electronic Archiving. *American Libraries,* 31(11):46–48.

[4] T. Fawcett and F. Provost, 1997. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3):291–316.

[5] Michelle Finley, 2000. Smart Methods to Spot Fraud. *Wired News*, April.

[6] S. Forrest and M. Mitchell, 1993. What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. *Machine Learning*, 13:285–319.

[7] K. Guthrie, 2001. Archiving in the Digital Age: There's a Will, But Is There a Way? *EDUCAUSE Review,* 36(6):56–65, November/December.

[8] A. Jain, et al, 1996. Artificial Neural Networks: A Tutorial. *IEEE Computer*, 29(3):31–44.

[9] T. Mitchell, 1997. *Machine Learning*. McGraw-Hill.

[10] T. Senator et al, 1995. The Financial Crimes Enforcement Network AI System. *AI Magazine*, 16(4):21–39.