SANDIA REPORT
SAND2013-5511
Unlimited Release
July 2013

Omen: Identifying Potential Spear-Phishing Targets Before the Email is Sent

Jeremy Daniel Wendt

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 6056900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods/asp?loc=7-4-0#online

SAND2013-5511
Unlimited Release
July 2013

Omen: Identifying Potential Spear-Phishing Targets Before the Email is Sent

Jeremy Daniel Wendt
Information Systems Analysis Center
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1027

Abstract

We present the results of a two year project focused on a common social engineering attack method called "spear phishing". In a spear phishing attack, the user receives an email with information specifically focused on the user. This email contains either a malware-laced attachment or a link to download the malware that has been disguised as a useful program. Spear phishing attacks have been one of the most effective avenues for attackers to gain initial entry into a target network.

This project focused on a proactive approach to spear phishing. To create an effective, user-specific spear phishing email, the attacker must research the intended recipient. We believe that much of the information used by the attacker is provided by the target organization's own external website. Thus when researching potential targets, the attacker leaves signs of his research in the webserver's logs. We created tools and visualizations to improve cybersecurity analysts' abilities to quickly understand a visitor's visit patterns and interests. Given these suspicious visitors and log-parsing tools, analysts can more quickly identify truly suspicious visitors, search for potential spear-phishing targeted users, and improve security around those users before the spear phishing email is sent.

Contents

List of Figures

1 Introduction

Spear phishing has become a standard technique for attackers to gain first entry into a targeted computer network. The RSA attack of 2011 that leaked critical security keys used by companies throughout the world started with a spear phishing attack [Rivner, 2011]. The United States White House confirmed in 2012 that – although no critical information was lost – a minor network was compromised as the result of a spear phishing attack [McCullough, 2012].

Part of spear phishing's strength is that it attacks one of the weakest elements in computer systems today – the human user. In a spear phishing attack, a targeted user is tricked into following a link to a corrupted webpage or opening a malware-laced "useful" program they received via email. Spear phishing belongs to an old, well known, and well researched field known as social engineering [Cialdini, 1993]. Due to social engineering techniques' long history of success, it is hard to believe that spear phishing can be solved simply by teaching the users: At some point, the users will relax their guard, revert to basic human responses, and fall victim to an attack. Moreover, specially trained analysts cannot monitor all email traffic at all times to identify suspicious emails. And, since spear phishing emails are specially crafted for their target, computer algorithms have not yet been able to consistently identify spear phishing emails.

These three techniques (increased self-guarding, increased analyst guarding, and automated guarding via computer algorithms) all fail for different, but related reasons:

- Self guarding fails because the human user must be ever vigilant at all times.
- Analyst guarding fails because analysts can't watch all email traffic for all users – and even if there were enough analysts, they would quickly tire.
- Computer algorithms have insufficient context (at present) to identify all factors that identify suspicious emails.

However, we believe that each of these spear-phishing guards could be improved for specific users for brief periods of time. The difficulty is in identifying the period of time and the specific users to guard. After providing definitions for a few key terms, we will describe why we believe spear phishing attacks can be predicted.

Definitions: *Phishing* is a technique where an attacker sends a uniform malware-laced email to a large number of recipients, hoping that some small percentage will compromise their machines. Phishing can be compared to its intentional homonym "fishing" in that the phisher is casting a wide net, and then pulling it in to see what/who he caught. *Spear phishing* differs in that the attacker targets a specific victim and is willing to spend the extra time necessary to catch him.

Spear phishing is often the first entry into a target network. Although the targeted information may not be stored on the machine where the email is opened, once the attacker has control of a computer inside the network defenses, he can more easily find and subvert useful machines.

In preparation for a spear phishing attack, the attacker needs to identify and learn about a target individual – his interests, coworkers, etc. As part of this research, the attacker likely visits webpages provided by the target's employer's external website. A *webpage* is an HTML document

combined with various supporting documents (images, layout definition files, support code, etc.). Each webpage can *link* – or refer – to other related web pages, and the attacker can follow these links to gather more information.

The attacker's web browser (a specialized program for viewing webpages; e.g., Internet Explorer, or Google Chrome) downloads the various documents from the company's web server. The *web server* is a computer set aside to serve requests for web pages. Each request is dutifully recorded in the web server's *log*. The log contains several details about each request – who made the request, at what time, what was requested, how the server responded, etc.

Predicting attacks: We believe predicting spear phishing attacks is possible due to a few characteristics unique to spear phishing. Figure 1 provides an overview of a spear phishing attack. As described above, the attacker's background research included visits to the target organization's webpage. These visits provide *warning data* to analysts. Moreover, the time the attacker must spend crafting the email – combined with the time before the targeted user notices, reads, and acts on the email – provide *response time* for the analyst to respond before the attack succeeds. These two features – warning data and response time – create a sort of race: If the analyst can find the right warning data faster than the spear phishing email is sent and acted on, spear phishing attacks can be defeated.

So, our task is two-fold:

1. Identify and highlight suspicious patterns in the logs.

2. Provide the analyst with helpful supplementary information so that the analyst can quickly and effectively differentiate benign from malicious visits.

We approach this task using two complementary techniques. Statistics can be used to separate different visitor patterns, prioritize items within the same pattern, and summarize groups. Visualizations can be used to summarize, highlight, and organize results. By using both statistics and visualization, we help analysts in their race to identify potential spear phishing targets before they receive the spear phishing email.

Organization: The remainder of this section gives an overview of previous work on analyzing web server logs. Section 2 describes the values that are available in input logs. Section 3 provides the precise criteria we use to identify different visitors in logs. Section 4 describes each statistical technique we use to help analysts find and understand suspicious visit patterns. Section 5 describes how we applied standard visualization techniques to summarize and organize results. Section 6 lists several approaches we took that did not provide results for this work, but which may prove useful for future developments in this or other areas. In section 7, we discuss conclusions, lessons learned, and propose further work.

1.1 Previous work

This section is not a full analysis of all papers that have used web logs for mining. For more complete overviews, see [Srivastava et al., 2000] or [Eirinaki and Vazirgiannis, 2003]. From what we could find, no one else is analyzing web server logs to identify potential spear phishing targets.

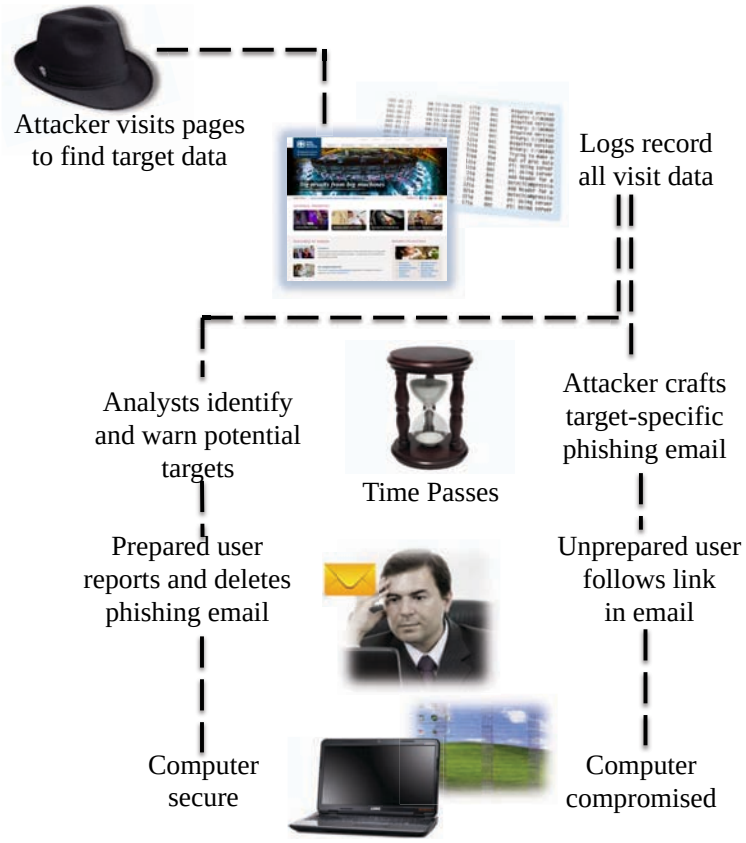

Figure 1: **Spear Phishing Overview**: A spear phishing attack requires the attacker to collect background data on his target. The attacker likely visits the target organization's website collecting that data. The attacker then needs time to craft the spear-phishing email, including any necessary modifications for the malware-laced attachment. After the spear-phishing email is sent, considerable time may still pass before the target reads the email, opens the attachment and compromises his machine (right-side path). Because the attacker's visits were stored in the web server logs, the target company's analyst has data concerning the attacker's visits. Moreover, because the attacker spends time crafting the spear-phishing email, and the recipient may not read the email for some time, the analyst also has time to identify potential spear phishing targets. If the analyst can identify potential targets and increase security around them, the spear-phishing email could be identified and deleted – keeping the computer secure (left-side path).

One common use of web server logs is to improve web server performance. Almeida et al. analyzed their logs looking for spatial and temporal locality of requested pages to decrease cache misses for requested webpages [Almeida et al., 1996]. Similarly, Iyengar et al. analyzed usage trends on seasonally popular web servers to predict how future loads would affect hardware needs [Iyengar et al., 1999].

Another common web server log analysis motivation is to identify common usage patterns to improve website design – optimizing the common uses. Chen et al. looked at paths users took through websites and created string representations for these paths [Chen et al., 1996]. They looked for longest common substrings and proposed creating obvious links to the last elements on those substrings from the first – permitting users to skip all steps in the middle. Cooley et al. came up with three heuristics to separate "navigational pages" from "content pages" [Cooley et al., 1997]. They propose shortening paths that go through multiple navigational pages before reaching the useful content pages.

2 Inputs

Web server logs contain one event per line. Each event contains some subset of the following data values (null values are inserted for unknown or unavailable values). Each of the following descriptions follows the same pattern: a brief descriptor in bold, more detailed description of value, parenthesized data type and format.

- **Timestamp**: Date/time in seconds since the epoch (floating point number).
- **Client IP**: IP address of the client machine requesting the resource (character string following IPv4 convention for IP addresses).
- **Client port**: Port number that the client machine is communicating on (integer).
- **Server IP**: IP address of the web server request sent to (character string following IPv4 convention for IP addresses).
- **Server port**: Port number that the server machine receives the request on (integer).
- **HTTP method**: The HTTP method requested (character string – one of GET, HEAD, POST, PROPFIND, OPTIONS, PUT, REPORT, LOCK, CONNECT, null).
- **Server name**: The hostname of the web server that contains the requested resource (character string following hostname convention (e.g., hostname.domain.org)).
- **Requested resource**: The path to the requested resource/file (character string with slashes separating directories).
- **Refer string**: The URL of the page that contained the link used to find this resource (character string – can be null if no link followed for this resource).
- **User-agent string (UAS)**: The client program's self description (character string – more details below).
- **HTTP request size**: The size of the client's HTTP request's body (integer – in bytes).
- **HTTP response size**: The size of the server's HTTP response's body (integer – in bytes).
- **HTTP status**: The status code for the server's response (integer – see [Wikipedia, 2012a] for possible values).
- **Info message**: Further details (character string – almost always null).
- **X-forwarded for (XFF)**: Details about any address-translating proxy servers (character string – more details below).
- **Content type**: Brief description of the type of file transmitted (character string).
- **Filename**: Filename details (character string – almost always null).

- **MIME type**: Brief description of the type of file transmitted – often different than content type (character string).
- **MD5 hash**: Hash of the data transmitted (character string of hex value – almost always null).
- **Extraction file**: Unknown (always null).

UAS overview: The user-agent string (UAS) is a string provided by the client to identify itself to the web server. As a generic string, it can be whatever the client may wish. However, most web browsers' UASs follow this pattern [Wikipedia, 2012b]: `Mozilla/[version] ([system and browser information]) [platform] ([platform details]) [extensions]`[1].

This pattern comes with caveats. When analyzing the UASs present in our logs, I found that 58% of the UASs only contained the first parenthesis pairing, and another 3% of browser-based UASs didn't match this pattern at all. Also, while browser UASs are somewhat standardized, crawler UASs are not.

Finally, it is important to note that the UAS is completely under the control of the client. There is no way for our web servers to verify the UAS represents the client program's settings. Also, there is no penalty for false UAS data: Many servers ignore the UAS (other than to log it), and those that use it often only check to see if the UAS represents a crawler or a browser client.

XFF overview: The X-forwarded for (XFF) field describes any client-IP address changes that occurred as the request was sent over the Internet [Wikipedia, 2012c]. Depending on settings, the client's IP address may change when a data packet crosses proxies, or NAT servers. When the IP address in the client IP field is changed during network routing, the intermediary appends the old client IP address to the list stored in the XFF field.

Proper parsing of the XFF field is critical for how we identify distinct visitors (Section 3). The general format for XFF strings is as follows: `X-Forwarded-For: client, proxy1, proxy2`. However, we found the following factors critical in parsing these strings:

- Not all non-null XFF values contain the string "X-FORWARDED-FOR ->". If it doesn't contain that string, we treat the field as null.
- Not all that contain "X-FORWARDED-FOR ->" contain only the XFF information. To remove others, we performed the following:
  - Strip any characters from the string that precede the first "X-FORWARDED-FOR ->" string.
  - If it contains "VIA ->", "PROXY-CONNECTION ->", or "CLIENT-IP ->", strip that and all characters after it from the "proxied" string.
- Often, the XFF is delimited by "\x2c" (the unicode representation for ','): We split on both.

[1]The square brackets are added only to differentiate actual standard text/punctuation from my descriptors. UASs don't generally contain square brackets.

	March logs	October logs
Start date	5 March 2012	23 October 2012
Day 1	675,126	742,571
Day 2	691,718	820,866
Day 3	510,022	789,495
Day 4	491,059	674,900
Day 5	508,349	505,209*
Day 6	322,295*	539,087*
Day 7	308,515*	1,280,645
Day 8	494,303	2,163,887
Day 9	614,857	752,447
Day 10	506,462	741,927
Day 11	649,708	644,751
Day 12	429,692	494,657*
Day 13	335,308*	31*
Day 14	417,045*	822,956
Day 15		822,956
Day 16		681,967
Day 17		665,699
Total	6,955,429	12,320,988

Table 1: **Number of entries by date**: The number of entries in each log by date. Entries marked with an asterisk (*) were weekends, and generally lower visit dates. The logs for 4 November 2012 (Day 13, column 2) were lost excepting a few entries at the very beginning and end of the day.

- Sometimes the "X-FORWARDED-FOR ->" string is repeated before new IP addresses. We strip out the repeated string.
- The "IP address" is sometimes one of the following: "none", "unknown", or "UWC". we ignore these.
- IPv6 addresses can show up in this list.

Much like the UAS, the XFF field is an unverifiable, client-provided field with no penalty for providing false XFF data. We are unaware of any way that web servers use XFF data.

Web server logs: Thus far, we have described a single entry in a log. Web server logs contain an individual entry for each item requested by a web-based client. Thus, when downloading a single web page, the client must separately request the HTML, any images, layout files, code, etc. Each of those requested items will leave a separate entry in the log – and these entries are likely intermixed with entries for hundreds of requests from other clients. The remainder of this section provides overview statistics from the logs used for the remainder of this report.

In performing this analysis, we used logs from all traffic to Sandia's web servers from clients outside Sandia. Specifically, we used logs from two periods: 5 March 2012 to 18 March 2012 (called "March logs" hereafter), and 23 October 2012 to 7 November 2012 (called "October logs" hereafter). Table 1 provides an overview of the number of visits seen on each day. Weekend days often had fewer visits than weekdays. This indicates that much of our traffic is business-related. The drastic increase in

	March logs	October logs
GET	98.5%	96.5%
HEAD	0.7%	0.5%
POST	0.4%	2.8%
PROPFIND	0.3%	0.1%
OPTIONS	0.1%	0.04%
PUT	0.01%	0.07%
REPORT	<0.01%	0%
(empty)	<0.01%	<0.01%
LOCK	<0.01%	0%
CONNECT	<0.01%	<0.01%
TRACE	0%	<0.01%

Table 2: **HTTP Command Usage**: Relative usage of each HTTP command in the two logs.

	March logs	October logs
Google	96.2%	93.3%
Bing	1.6%	2.4%
Yahoo Search	1.3%	1.6%
Baidu	0.4%	1.5%
Yandex	0.3%	0.9%
Ask	0.3%	0.25%
Duck Duck Go	0.01%	0.01%
Blekko	0.01%	<0.01%
Vadlo	<0.01%	<0.01%
Chacha	0%	<0.01%
TOTAL SEARCHES	296,841	212,734

Table 3: **Search Engines Used**: Relative percent of different search engines used by visitors (as identified in refer strings).

the number of visitors on Days 7 and 8 of the October logs was due to a misconfigured external machine requesting a huge number of pages.

HTTP method: Although HTTP provides several methods, GET is by far the most common (see Table 2). Traditional webpages interact almost exclusively through GET. However, a new web-programming style (REST) uses HTTP to transfer requests and data. This has led to an increase in the use of other methods.

Search engines: When a visitor follows a link on a search engine, their browser generally includes the URL from the search engine in the refer string. This allows us to estimate search engine usage by those visiting our webpage (see Table 3). Another benefit from this refer string is that often the refer string contains the original search terms. We describe how we leverage these search terms in creating visitor summaries in Section 5.

Most common websites: Sandia maintains around 260 distinct web sites (where a site is defined as a different server name; e.g., "www.sandia.gov" is distinct from "trilinos.sandia.gov"). These 260 web sites are hosted by over 50 different machines (as judged by distinct IPs). However, these different sites receive a drastically different number of hits. Table 4 lists the top eight most visited

	March logs	October logs
www.sandia.gov	36.0%	34.5%
mems.sandia.gov	22.0%	6.5%
lammps.sandia.gov	6.8%	7.1%
energy.sandia.gov	6.2%	6.9%
prod.sandia.gov	5.0%	5.4%
trilinos.sandia.gov	2.8%	10.4%
photovoltaics.sandia.gov	2.6%	0.8%
anywhere.sandia.gov	2.4%	1.4%
cubit.sandia.gov	0.01%	18.5%

Table 4: **Most commonly visited servers**: Relative percent that different servers were requested by visitors (top eight from each period shown; cubit and photovoltaics were only in the top eight during one period).

web sites over the two periods. While www.sandia.gov is the most commonly visited site during both periods, the other sites trade order, and only seven of the top eight are the same in both periods.

3 Identifying Visitors

The anonymity of the Internet increases the difficulty of analyzing web server logs. As described in the last section, the logs do not contain specific user information: We don't have the user's name, pseudonym, physical location, machine-specific data, etc. We only have two real client machine identifiers: IP address, and UAS. However, as was pointed out in the last section, the IP can be altered in transit, with the XFF storing intermediary IP addresses. If the XFF contains any IP addresses, the IP address reported in the client IP field is not the originating IP address – the first IP address in the XFF list is.

Identifying visitors is made more complex by private IPv4 spaces. Due to insufficient IPv4 IP addresses, many local networks use private IP addresses and then use network address translation (NAT) at the local network's boundary so that only one public IPv4 address is used. Therefore, the first entry in the XFF field may be a private IPv4 address – an address that can be reused behind NAT servers by countless computers. Furthermore, the first public IPv4 address in the XFF (or client IP field) is likely a NAT server with hundreds of private-IP client machines behind it.

The IP address becomes less reliable still because of Dynamic Host Configuration Protocol (DHCP). DHCP assigns an IP address to a machine when it first enters the network. After a specified duration with no traffic from that IP, DHCP will re-assign that IP to a new machine. Thus, two different machines may provide the same IP address at different times.

A similar issue arises in IPv6 islands. Among other reasons, IPv6 addressing was created to overcome IPv4's limited IP address space. The transition between IPv4 and IPv6 has been on-going for years and is likely to continue for many more years. Therefore, the transition model provides for IPv6 islands. Within an IPv6 island, all machines use IPv6 addresses. At the boundary to the rest of the Internet, the IPv6 addresses are replaced by IPv4 addresses. Similar to NAT servers and private IPv4 spaces, a single IPv4 address can hide hundreds of different client machines.

So, IP address alone is not sufficient to differentiate between machines: NAT and IPv6 islands hide hundreds of distinct IPs behind one IPv4 address. Private IP spaces reuse the same IP addresses behind each NAT server. DHCP reassigns the same IP address to different machines.

User-agent strings (UAS) are no more perfectly distinguishing. The UAS is a client-program-defined string provided to the web server. For crawlers, it can include crawler program name and version information. For browsers, it often contains browser version and some client operating system information. Moreover, due to browser settings, crawler programming, or network settings, the UAS can be null. Whenever two identical crawler programs or two browsers with identical settings and operating system make requests, their UASs are the same.

To overcome the possible collisions from IP addresses behind NAT servers, we use both the private and public IP addresses together. Thus, at any given time, only one machine should reside behind the public NAT server's IP with that specific private IP address. To overcome DHCP's tendency to change IP addresses for clients over time, we also add the UAS to the identifier.

In brief, our identifier is as follows: private IP[2]/public IP/UAS.

There are some important notes about this identification system:

[2]If no private IP exists, this is omitted

- This method limits possible overlaps between different clients appearing under the same identifier, but increases the likelihood that the same client may appear as multiple distict visitors (if his IP address is changed by DHCP, or if a configurations change results in altered UAS).

- It is still possible for two clients to overlap. However, to do so, they must obtain the same IP address (due to DHCP shifting the IP from the old machine to the new one) *and* be using an identically configured client program. In the case of browsers, we decided this was an acceptable risk.

4 Statistics

As mentioned in Section 1, we are trying to help the analyst find suspicious patterns faster than the attacker can craft a spear phishing email, send it, and the user follows a link. However, as shown in Table 1, on any given day, around half a million visits are made to our webpages. Therefore, we must separate the log entries into different groups, hide less relevant data, and highlight suspicious patterns. In this section, we describe techniques we used to perform these functions.

4.1 Crawler or Browser

Both human-driven browser traffic and script-based crawler traffic show up as entries in our log files. Often, the analyst is only interested in one of those groups. Furthermore, crawlers and browsers generally request content for different reasons. Crawlers can serve a few different purposes: indexing a website for a search engine (googlebot); downloading entire portions of a website for off-line viewing (wget); one-off programs performing a specific task; etc. Browsers are generally driven by a human user typing URLs or search strings, or clicking on links. These different purposes lead to different visit behaviors. One group's behavior can mask intricate patterns in the other group.

Therefore, we want to separate human-driven browser traffic from programmatic crawler traffic. The user agent string (UAS) often includes sufficient information to differentiate crawlers from browsers. However, since the UAS is provided by the client, we can't simply trust it: A client can lie. So even though we begin by grouping the visitors based on UAS, we look for behavioral patterns that reliably separate the groups.

Splitting on UAS results in three self-identified groups – browsers, crawlers, and null (those who didn't provide a UAS).

The following analyses were performed against the March logs. To ensure sufficient per-visitor data, the analyses examined only visitors who had left 20 log entries.

Visit characteristic – content type: Browsers want to display a completed webpage to a human user. This requires downloading all files required for the webpage – HTML, image files, layout files, code files, etc. On many webpages, the supporting files (images, layout, code, etc.) can outnumber the HTML by more than ten-to-one. Moreover, for every webpage downloaded by a browser, the browser will often request all items. This is done to ensure the latest content is visible.

Programmatic crawlers' download patterns depend on their purpose. If they are indexing for search (Google), then they primarily need the words on the page. In this case, HTML would interest them most strongly – nearly all indexable terms are in the HTML. If they are downloading portions for off-line visits (wget), then they will only need one copy of each item. Therefore, they won't re-download an image file, code file, or layout file that was used on a previously downloaded page.

Thus, browsers download more non-HTML on average than crawlers. We created a metric based on this characteristic as follows:

$$\%\text{HTML} = \frac{\text{numHtml}}{\text{numTotal}} \tag{1}$$

The histograms for the three UAS-defined groups are shown in Figure 2.

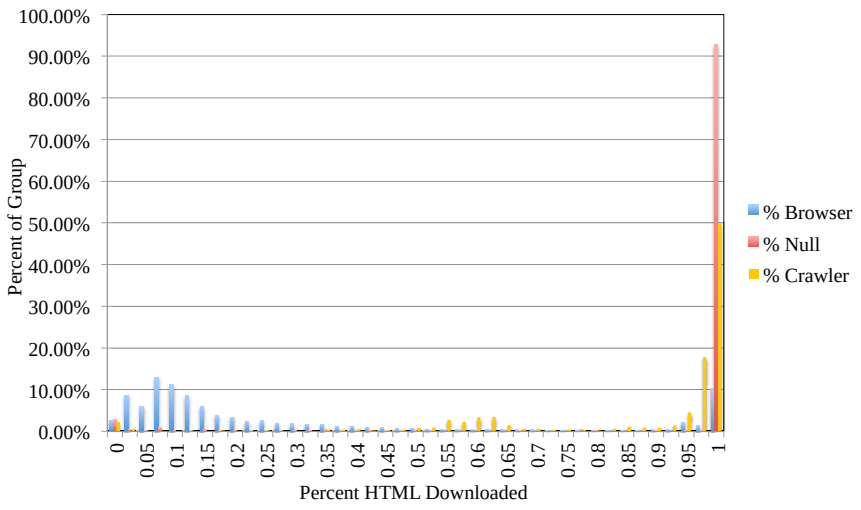

Figure 2: **Crawler or Browser by Content Type**: All downloads were divided into two types: HTML and not-HTML. A visitors' score on this histogram was determined by dividing their number of HTML visits by their total number of visits. Most browser-UAS visitors have a score below 0.5. Most crawler-UAS visitors have a score above 0.5. Nearly all null-UAS visitors have a score of 1.

- More than 90% of null-UAS visitors downloaded nothing but HTML content. The only other significant group of null-UAS visitors are those that downloaded no HTML content.

- Approximately half of crawler-UAS visitors downloaded only HTML content, with a slow decline through the mid-80% HTML content range. There is a small hump (<20% of all crawler-UAS visitors) between 55% and 70% HTML. There is a small group of crawler-UAS visitors that downloaded only non-HTML content.

- Most browser-UAS visitors sit between 2% and 50% HTML content (with a heavy right skew). There is a small group of browser-UAS visitors that downloaded only non-HTML content. There are a significant number of browser-UAS visitors that show up in 90+% HTML range.

Most of the results in Figure 2 are what we would expect: Most crawlers download mostly HTML; most browsers download a heavy mix of non-HTML. This also confirms suspicions that most null-UAS are likely crawlers: They almost exclusively download HTML. However, there are some interesting results. First, the hump of crawler-UAS visitors around 60% are likely crawlers that need the layout files (e.g., wget) but only need them once instead of for every page. Second is the fact that all three visitor types have a small amount of visitors at exclusively non-HTML (0% on the x-axis). We have no good explanation for these visitors except to believe them likely distributed crawlers: How else could they visit at least 20 times over two weeks and never retrieve a single HTML page? The third group is perhaps the most interesting – the browser-UAS visitors at more than 90% HTML downloaded. The browser-UAS visitors all drop off around 50% HTML downloaded but then suddenly appear within the crawlers. To further categorize these users, we developed a second metric.

Visit characteristic – burstiness: Programmatic crawlers can generate requests quickly enough to tie up a web server's resources – creating a DoS attack. Any crawler that makes so many requests will be blacklisted by web server administrators – blocking them from downloading from

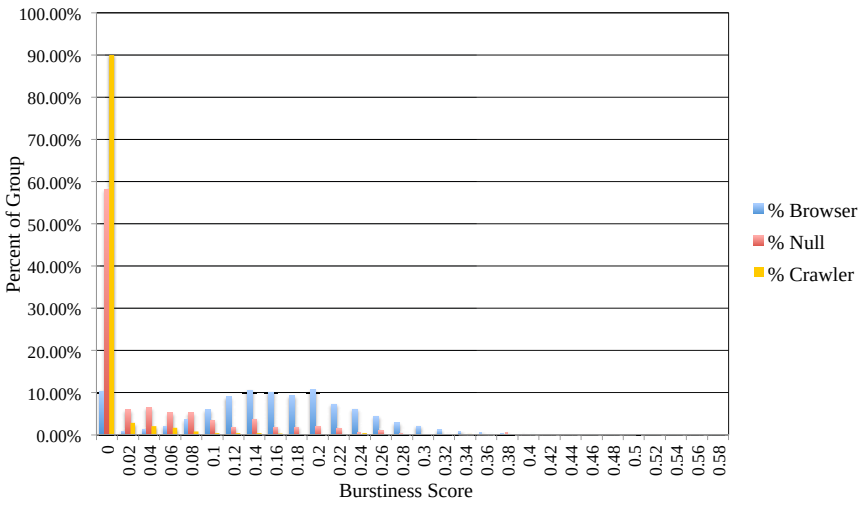

Figure 3: **Crawler or Browser by Burstiness**: We counted the number of visits by the same visitor within a M second window. If there were more than N visits in that window, we counted that as a burst. The window advanced by M/2 seconds and counting continued: The windows overlapped by half. A visitor's "Burstiness Score" (x-axis) was the ratio of the number of bursts to the number of visits. Nearly all crawler-UAS visitors had no bursts. More than half of null-UAS visitors had no bursts. Most browser-UAS visitors had some number of bursts. Their hump is centered around a 0.18 score.

the servers again. To avoid being blacklisted, programmers create their crawlers to make requests below a certain rate. During the delays between same-server requests, crawlers make requests of other servers or stall their code. This leads to crawlers generally requesting only one item from a server at any given time.

Browsers follow a very different pattern. For a browser, the goal is to present the user with a complete webpage as quickly as possible. Therefore, a browser requests all non-HTML support files in rapid succession to the original HTML. This browser burst does not effect overall web server performance because the browser only bursts until one page's content is downloaded. The browser then sits idle for seconds to minutes while the user consumes that page's contents. Only after the user issues the next content request will the browser again burst.

We created a metric based on this characteristic as follows. We count the number of visits made by a visitor within a M second window. If the visitor makes more than N visits, we count a burst. The window advances by M/2 seconds and counting continues. A visitor's "Burstiness Score" is the ratio of bursts to number of visits. The histograms for the three UAS-defined groups are shown in Figure 3.

- Ninety percent of crawler-UAS visitors had no bursts. This is followed by a rapid drop off for the remaining crawler-UAS visitors – with almost none receiving a score above 0.1.

- More than half of null-UAS visitors had no bursts. This is followed by a slow drop off for the remaining null-UAS visitors – with almost none receiving a score above 0.28.

- Most browser-UAS visitors had some number of bursts with almost all in a hump centered around 0.18. However, 10% of the browser-UAS visitors had no bursts.

Most of the results in Figure 3 are what we expected: Most browsers exhibit some amount of burstiness; most crawlers have no bursts – including the null-UAS visitors. This further confirms our suspicions that null-UAS visitors are crawlers. The most interesting result is that approximately 10% of browser-UAS visitors had no bursts – generally a crawler feature. If these browser-UAS visitors are the same that exhibited crawler-like features in the other metric, this would further confirm our suspicions that they are actually crawlers. This metric provides less separation than the Percent HTML Metric: The null-UAS visitors spread well within the browser-UAS visitors' burstiness hump. When combining the two metrics, we hope to get better separation than we could with either individually.

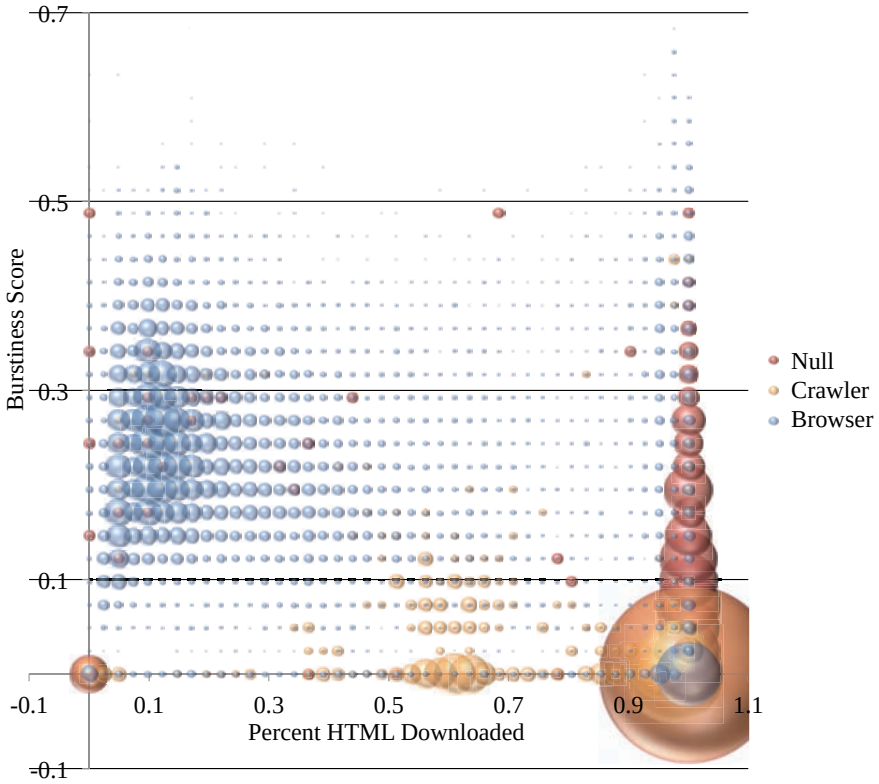

Figure 4: **Crawler or Browser – Both Metrics**: This combines both the Percent Content Type and the Burstiness Metrics in one plot. The x-axis reports the Percent Content Type. The y-axis reports the Burstiness. The size of a colored circle at any point on the graph indicates the relative percent of visitors that fell into that location for both scores. Note that both null-UAS and crawler-UAS visitors have their largest groups at 100% HTML and no bursts. Null-UAS visitors spread mostly up the y-axis from there. Crawler-UAS visitors spread mostly down the x-axis from there. Most browser-UAS visitors are in an island in the upper-left side of the graph. After almost no browser-UAS visitors, a significant number show up at the right side of the graph.

Combined metrics: We see the two metrics combined in Figure 4. The upper-left section of the graph is dominated by browser-UAS visitors. These visitors demonstrate some number of bursts, and download a reasonable percentage of non-HTML items. Their cohesive, Gaussian-like shape indicates that they are generally a single group.

The bottom-right corner point has two nearly identical-sized circles for crawler-UAS and null-UAS visitors. This single data point accounts for approximately half of each group. There is a much smaller co-located circle for some number of browser-UAS visitors. This point represents 100%

HTML downloaded with no bursts at all – typical, well behaved indexing crawler activity.

The crawler-UAS visitors mostly continue along the x-axis on the graph. This represents visitors who downloaded some number of non-HTML websites but still with no bursts. The crawler-UAS visitors along this axis tend mostly toward the right side of the axis – more HTML than non-HTML. We believe these visitors are crawlers similar to wget – crawlers that pull down an entire website for off-line viewing. The reason these visitors do not need as much non-HTML content as browsers is that many pages within a single website share non-HTML content – the same icons or images shared across multiple pages, the same layout file may be used by all. These off-line storage crawlers need download those files only once.

With a few exceptions, the null-UAS visitors along the x-axis are mostly at either extreme – 100% or 0% HTML downloaded.

Excepting at the extremes, the browser-UAS visitors along the x-axis are very uniformly distributed.

The right-most Y-axis is dominated by null-UAS visitors. Along the same axis (and immediately near it) there are a significant number of browser-UAS visitors. They appear different from the other browser-UAS visitors in two ways:

1. Their burstiness range is much higher and continuous than the accepted browser hump in the upper-left section.

2. They are clearly separated from the accepted browser hump on the percent-HTML axis.

There are four remaining minor areas to discuss:

- The left y-axis (0% HTML) is mostly empty. There are a very few null-UAS vistors on this axis mixed with a number of browser-UAS vistors.

- Toward the middle-right of the x-axis, toward the bottom, there are some number of crawler-UAS visitors. Some of these visitors overlap with the farthest reaches of the browser hump.

- There are a handfull of null-UAS visitors within the accepted-browser hump. There is a good (albeit ethically bad) reason for a human-driven browser to not send UAS information to servers: cheating on membership-protected content. Various news websites (nytimes.com, wsj.com, etc.) derive revenue by permitting access to their content to only those users who maintain a membership. Users are password authenticated before they are allowed content. However, these websites want their content listed on search engines. This would require either providing all search engines with passwords, or permitting search engines to retrieve content without passwords. These websites have chosen the latter option, but have only the UAS to determine if a visitor is a browser or a crawler.

- There are regions on this graph with no visitor data from the March logs. This does not mean no visitor could arrive in those spaces – only that none did during this period.

Results: No matter where you divide this space between "browser" and "crawler", some visitors will be marked differently than their UAS indicated. A division we chose identifies 98.6% of crawlers-UAS visitors as crawlers, 96.8% of null-UAS visitors as crawlers, and 81.8% of browser-UAS visitors as browsers. These numbers are improved by the following:

	March logs	October logs
Identified Crawlers	10.3K	10.7K
Identified Browsers	38.6K	47.4K
Too Few Visits to Identify	166K	161K

Table 5: **Number of visitors assigned to each group**: The number of visitors assigned to each group by the browser vs. crawler analysis. Most visitors had too few visits (less than 20) to be statistically described for the analysis.

	March logs	October logs
Visits by Crawlers	3.2M	5.8M
Visits by Browsers	3.0M	5.8M
Visits by Too Few Visits	0.8M	0.7M

Table 6: **Number of visits by each group**: The number of visits by visitors assigned to each group. Even though the browser vs. crawler analysis can't identify most of the distinct visitors, it identifies the visitors for around 90% of the total visits in each period.

- Approximately 1% of null-UAS visitors are likely membership-protected content theives – improving null-UAS identification to ˜98%.
- We believe (to varying degrees of certainty) that 10-17% of the browser-UASs-marked-as-crawlers are actually crawlers – improving browser-UAS identification to ˜92-99% identification.

Table 5 shows the relative distribution of visitors as assigned by this analysis. In both periods, the number of identified crawlers was much smaller than the number of identified browsers – which in turn was much smaller than the number of visitors with too few visits to be statistically described. The large number of undefined visitors may make this work seem less useful. However, as shown in Table 6 that very large number of too-few-visits visitors are responsible for only 11% and 5% of the total visits in each period. Thus, the vast majority of visits to our webservers are now assigned to a visitor that we categorized as a crawler or a browser based on their visit patterns. The remaining few visits – if considered interesting – could be categorized solely by the UAS.

Furthermore, this analysis identifies many visitors that are likely lying in their UAS. These visitors may be more interesting to analysts than others assigned to the same group.

Finally, this analysis lets us do statistical analysis of browsers separately from that for crawlers with more confidence that the groups are truly separate. In the March analysis, more than 80% of the visitors identified as crawlers were identified as browsers by UAS. As shown comparing the relative size of the groups in Table 5 and their relative number of visits in Table 6, crawlers average 3-4 times more visits than browsers. If the browser-UAS crawlers had been left in the browser analysis, they would have clouded that difference considerably.

5 Interface

The previous section described statistics we performed to separate and decrease the quantity of data that analysts had to look at. However, even with the separation and pruning, the quantity of data presented was still far more than analysts could possibly examine. Thus, an analyst would still need to quickly separate (triage) the potentially troubling data from likely innocuous data. After identifying potentially troubling visitors, the analyst investiages those further. In this section, we describe user-interface and visualization tools developed to support this triage and investigation.

5.1 Triage Interface

We developed an interface for triaging visitors. We wanted to present significant useful information in a single interface that provides the analyst some detail information and some overview information. Our interface is shown in Figure 5. This interface is intended purely as a mock-up: If we were truly supporting analysts, we would integrate a similar UI into Splunk.

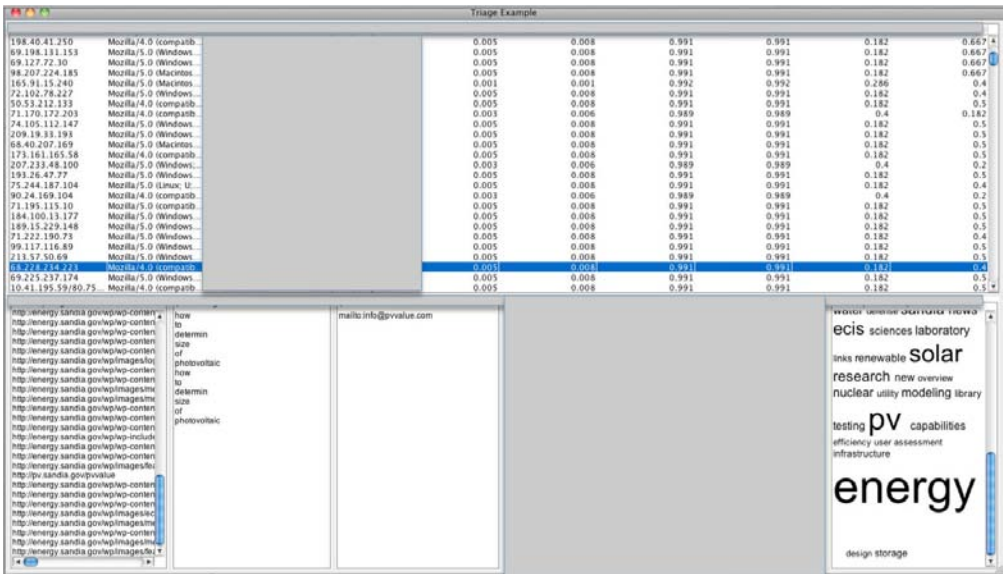

Figure 5: **Visitor Triage Interface**: We created a proof-of-concept user interface for the analyst to quickly triage the most interesting visitors. The top half of the screen presents information about the visitor – IP/UAS ids, and the visitor's scores on our various metrics. The bottom half of the screen provides more information on the visitor: URLs visited, any search terms used to come to Sandia's pages, any email addresses listed on the pages visited, and a word cloud overview of all visited pages.

This interface is split into a top and bottom half. The top half provides a list of the visitors. After providing identifying information for the visitor, various scores are shown. The bottom half of the screen provides further details for the selected visitor in the top half. Each of the entries is briefly described in its own paragraph below.

Visited links: The first element is the actual links visited. This allows a knowledgeable analyst to look for warning signs on their own.

Search terms: We also provide any search terms the visitor used immediately before visiting our webpages. If the visitor provides refer strings, then following a link to Sandia from most search engines helpfully provides the search terms used to find that page. These search terms provide insight into the specific thing the visitor was looking for that brought them to this page. As can be seen in Figure 5, the selected visitor was trying to solve the problem of how to determine the size of a photovoltaic.

Email addresses: We show any email addresses linked on pages that were visited. These are discovered by our crawler parsing any "mailto" links as email addresses. These can indicate what potential targets were found.

Word cloud description: Finally, we summarize the visited pages via a word cloud. Our crawler stores the number of occurrences of each word on each page. We combine these counts for all visited pages, and create a word cloud with the most common words' size relative to their count. This word cloud provides an immediate overview of the most important topics on these web pages.

5.2 Visualizing Websites

Monitoring visits to websites can be extremely difficult. Sandia maintains ˜160 distinct websites. Although many of those websites are very small (hundreds of files), others are incredibly large. In our crawls of Sandia's various websites, we found more than 16.8 million distinct URLs. The information's scope is massive.

Moreover, the information's underlying type further complicates the difficulty. URLs are plain text, but represent hierarchical information: A URL contains the domain name and the full directory structure for each file. While it is possible to quickly compare two neighboring URLs to identify how proximate they are to each other, the problem quickly becomes more complicated with more URLs.

We propose using TreeMaps as a visualization for presenting website overviews. TreeMaps were first proposed by Shneiderman as a way to visualize hierarchical harddrive space [Shneiderman, 1992, Shneiderman, 2013]. TreeMaps allow for three distinguishing characteristics within a single image:

1. A rectangular element's size;

2. A rectangular element's color; and

3. Rectangles nested within rectangles present hierarchical information.

In creating our examples below, we use the Prefuse implementation of Bruls et al.'s "squarified" layout algorithm [Bruls et al., 1999, Pre, 2013]. We developed our own rendering and UI code to interface with the resulting TreeMaps.

Similar to their original use, we use the hierarchy to present file location within the website's directory structure. To help emphasize the hierarchies, we add a thicker line around higher-level directories. We vary from the original as we fix each underlying URL's rectangle to the same area (instead of varying by file size). This allows directory sizes to quickly indicate relative number of distinct URLs within them.

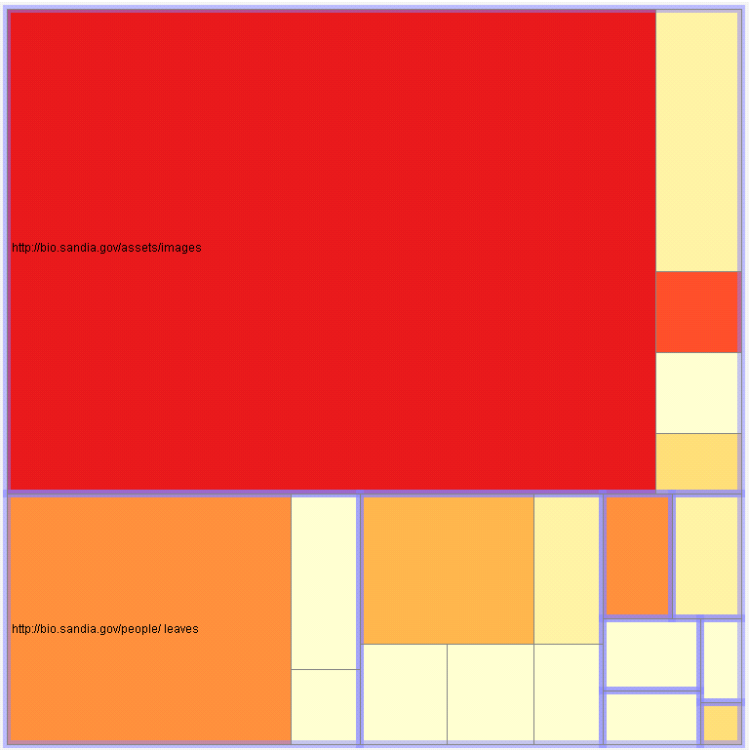

Figure 6: **TreeMap visualization of website**: We propose using TreeMaps to visualize websites. This TreeMap shows the directory structure of the bio.sandia.gov website. Each blue-outlined rectangle shows a top-level directory. Each gray-outlined, internal rectangle shows subdirectories within that directory. A directory's rectangular area is calculated by the number of files it contains. A directory's color could indicate several features. In this example, it shows the relative number of visits over a two week period (red being the most, light yellow the least).

We reserve color to represent any number of information visualizations. For instance, Figure 6 shows the TreeMap we formed from the bio.sandia.gov website. In that figure, the color represents the relative number of visitors to each directory during a two week period. Using this visualization, the analyst can quickly see that the directory bio.sandia.gov/assets/images/ (top left) is very large and is the most regularly visited. Moreover, the leaf files stored in the directory bio.sandia.gov/people/ (bottom left) is much less visited. Compare this to the un-labeled directory to the right of bio.sandia.gov/assets/images/ which is significantly smaller than the leaf files in people/, but is more often visited.

Color could be used to represent many factors:

- number of visitors,
- file type,
- file size,
- number of emails on a page,
- relevance to a user-specified query.

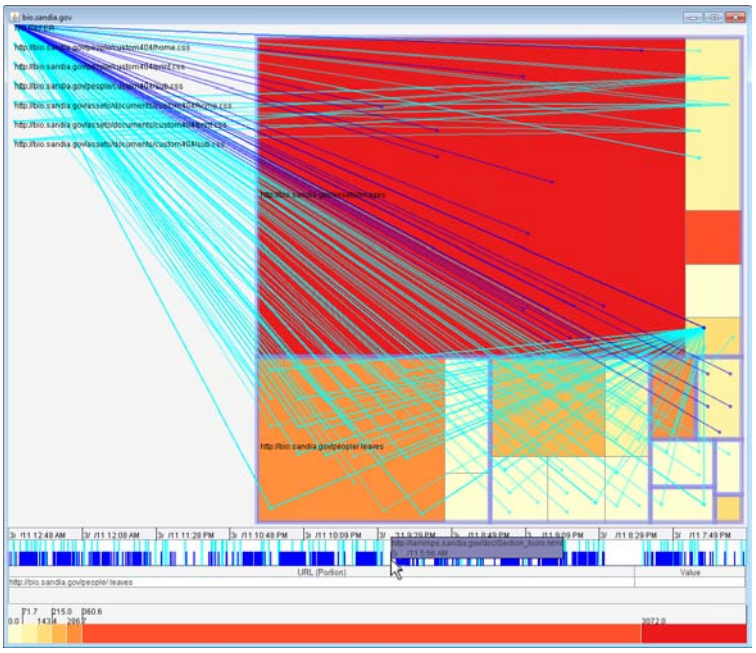

Figure 7: **TreeMap user interface**: We integrated our TreeMap visualization into a fully interactive user interface. The user can overlay different user's visit patterns (two visitors shown here in purple and light blue), interact with the TreeMap, and alter color divisions for the TreeMap coloring. Hovering over objects provides further information via tooltip.

We integrated this visualization into a tool for investigating visitors' visit patterns to Sandia's webspace (Figure 7). As shown in the figure, the user can add specific users' visits to the TreeMap. Although it would take some time to manually inspect each URL from these two visitors to identify them as nearly all different, this visualization shows it immediately. Moreover, it is quickly apparent that the purple visitor followed more common visit habits (heavily visiting the red rectangle, lightly visiting others), while the light blue heavily visited very rarely visited areas.

The UI also provides considerable secondary information for the analyst. Below the TreeMap, a timeline shows when each visitor made any visits to Sandia's websites. Hovering over any of these visits brings up a tool tip that provides further details about that visit. The section below that is updated constantly by the location of the mouse over the TreeMap – providing the directory's name (even if the rectangle is too small for a label) and the exact number of visits. Below that, a legend provides the boundaries for what values are represented by each color in the TreeMap.

This UI is also very interactive. The user can zoom and pan in the TreeMap to focus on smaller rectangles. When a rectangle's size becomes large enough due to zooming, labels become visible. The user can also alter the hierarchy level displayed. The timeline supports zooming and panning. The legend boundaries can be dragged – with the TreeMap color interactively updating to the new colors.

6 Future Work

As with any research, this research inspired more new ideas than those that could be explored in one project.

Purpose graphs: Much of this work was focused on providing quick, intuition-enhancing descriptors for websites and visitors. One of our early ideas that did not provide useful results fast enough we called "purpose graphs". The idea is to provide a quick summary of a website or a visitor based on all visitors' visit patterns.

We believe most visitors go to a website for a specific purpose. Therefore, the various webpages visited by any given visitor should further that visitor's purpose. To represent this between-webpage similarity, we create a weighted-edge graph of all of the pages visited by browser-based visitors. For each browser-based visitor's entry, we add a node for the visited page and the refer-string page (if provided). If the entry contains a refer string, we add an edge from the refer to the visited page. If the edge already exists, we increase its weight.

After creating the full graph from all visitors' visits, we have a weighted graph that should be a sub-graph of Sandia's full websites. The weights assigned to the edges indicate how heavily used those edges were. Thus high-weight edges should represent a pair of pages that more often help visitors further their purpose. We call this weighted graph the "purpose graph" because the weights should indicate how similar two page's are in purpose.

We believe this graph could be used for various purposes. Unfortunately, we were unable to follow through on any of these completely:

- We believe that a specific visitor's visits could be supplemented by the information in the purpose graph. The relative strength of edges followed by this visitor could indicate how much this visitor matches common patterns. Such a statistic could allow for visitor clustering beyond the (much simpler) bot-vs.-browser analysis described in Subsection 4.1.

- We believe webpages could be clustered by the purpose graph's weighted edges to provide a logical organization of the overall website – beyond the domain-name/directory-structure organization stored on the website. Unlike a clustering based on the full graph, this graph is based on which links users actually felt were important enough to follow. For instance, a considerable number of pages on our website contain a link to the main Sandia page (www.sandia.gov/index.html). Clustering on that graph would indicate that all of those pages are highly related to the main page. However, we would assert that the only relationship that usually exists is that the pages are built on a standard template that includes a link back to the main page. By instead basing the clustering on the links users followed, we can decrease the weight assigned to infrequently used template links.

Machine learning clustering: For part of our visitor-categorization work, we came up with as many characteristics we could for each user that visited our websites. We are uncertain how many of these are generally useful. For each unique visitor (see Section 3), we generated the following:

- **Number of visits**

- **Average download size** – The average size of file downloaded.

- **Approximate location** – Latitude, longitude, and country code as derived from their IP address.
- **Acts like a crawler** – Based on the analysis in Subsection 4.1.
- **Lying** – If their UAS doesn't match our crawler-or-browser identification.
- **Percent HTML** – See Subsection 4.1.
- **Number of bursts** – See Subsection 4.1.
- **Downloaded robots.txt** – Polite crawlers will check robots.txt before downloading a file. Thus downloading it could indicate a crawler. (Although there are reasons for browsers to download it; and with distributed crawlers, only one need download it.)
- **Number of favicons downloaded** – Favicon.ico is a small image that a domain can provide for the browser to add next to the URL. Crawlers have less reason to download it than browsers (although we saw one visitor that downloaded only favicon.ico several times – could be a bot).
- **Between-visit statistics** – The visitor's normal time between visits, and the percent of his visits that followed the previous visit by less than 0.5 seconds, 15 seconds, 30 seconds, 60 seconds, and 120 seconds.
- **Visits by time-of-day** – Percentage of the visitor's visits that came during normal business hours, vs. not; percentage of visits split by 4-hour periods throughout the day.
- **Visits by day-of-the-week** – The percentage of the visitor's visits that fell on each day of the week.
- **Percent refers** – The percentage of the visitor's visits that had a refer string.
- **Percent response type** – The percentage of the visitor's visits that fell into each 100-value range of possible server responses (200 = OK, 404 = Not Found, etc.).
- **Most common server** – The two most common server names requested by the visitor (and what percent of the visits were made to each).
- **Most common from port** – The two most common ports the visitor made requests from (and what percent of the visits were made from each).
- **HTTP command usage** – The percent of the visitor's requests that used each HTTP method.

These statistics were calculated during early work on the Browser-vs.-Crawler Analysis (Subsection 4.1; note that only two of them were used in the final analysis). After completing that analysis, we wanted to find other ways this large set of features could be used. Unsupervised machine learning can be used to create clusters from a large body of data. Warren Davis and Danny Dunlavy have been working on a system that automatically identifies an optimal number of clusters and identifies the features that add the most to the clustering. They ran this dataset through their preliminary

system and found that this dataset partitioned into 26 groups.[3] Unfortunately, they were unable to do a more complete analysis before the funding and time ran out. They were unable to identify the most significant features, examine the resulting groups for meaning, etc.

Merging "distinct" visitors: The method we described in Section 3 for identifying visitors is conservative: It is unlikely to identify two distinct people as the same visitor. However, it also will almost certainly identify the same person as multiple visitors over time. Such can arise from any of the following:

- The person uses multiple machines (different IPs and UASs).
- The person uses multiple browsers on the same machine (different UASs).
- The person's browser or OS update versions (altering the UAS).
- A plugin for the browser changes or is added (altering the UAS).
- The person moves the computer to a new location (new IP assigned by DHCP).
- The computer routes all traffic through TOR (an anonymizing service that changes the apparent IP).

We had several ideas of things that could be attempted for each of these, but have not spent time examining any of them directly. The machine learning described above would hopefully group two "visitors" if they were both being driven by the same person using our webspace in similar ways (may help with any of the first five).

We wondered if we could look at timing and similarity of use to identify the second group. Since the requests come from the same machine, the IP would be the same in both cases – only the UAS would change. If the only difference between two visitors was the UAS, but they had similar other properties, we could consider them the same user.

We began some work in parsing UASs to extract much more useful information. This included finding their OS (with version), as much browser information as possible, etc. Unfortunately, this analysis had to be curtailed before real results could be found. The hope is that we could identify "logical" changes in OS, browser versions, or additional plugins and this could solve the third and fourth changes listed above.

We had an interesting idea about TOR, but no time to evaluate it: We would want to do a full analysis of TOR, but expect that it alters the user's IP (by routing traffic through a different machine) at random times. If this is true, then when the browser requests a new page, it should still send the refer string – but we will not have ever seen that visitor (IP/UAS pair) retrieve data at that location. After identifying such a situation, we could look at the most recent visitors to download the content in the refer string, and watch for further activity from them. Any of these previous visitors who never request anything again (within a reasonable period) could be the previous IP for the visitor that since requested the content with refer string content that he never retrieved.

[3]These groups are not necessarily distinct clusters. These are 26 planar-separated groups. This means that if one group were L-shaped, and another group entered the concavity of the L, the L would be divided into two groups.

Collapsing requests under the main HTML: We had hoped to decrease the number of pages that analysts would need to look at when looking through a visitor's visits. As has been mentioned previously, when a browser requests a page, it also requests several "supporting" documents (images, layout, code, etc.). As shown in Figure 2, browsers download from two-to-one through ten-to-one supporting-documents-to-HTML. If we can identify all of the supporting content resources for each HTML page, we could nest those pages under the "main" HTML page. We tried different ways of accomplishing this – with varied, but incomplete success.

Our first attempt was to leverage the actual HTML page to identify the supporting documents that would be downloaded. The thought is that the browser is able to parse this page and identify what supporting documents it needs, so we should be able to do the same. We ran into two problems that limit the usefulness of this approach: First, by the time we crawled the page, often the HTML had changed somewhat, so the supporting documents the HTML needed when we crawled were different than the supporting documents needed when the visitor visited. Second, many of the supporting documents are contained within code the browser executes (e.g., Javascript). Because executing such code within a crawler could leave the crawler open to vulnerabilities, the crawler we used does not support scripting languages. This means that even if we downloaded the same version of the HTML, there would be many supporting documents we could not identify.

Our second attempt was to look in the logs to identify temporal patterns that would indicate supporting documents that were automatically downloaded versus manually requested documents. The thought was that automatically downloaded pages should be requested very quickly after the main page – manually requested pages should follow much later. Moreover, we thought that the standard deviation of the time between requests across all visitors that requested the same pair of files should be very low for automatically downloaded vs. manually requested. Unfortunately, when we computed these values for all file pairs that were requested by multiple users (a single data-point per pair of files), we saw no clear distinction between two groups (Figure 8). Surprisingly, the standard deviation started going up very rapidly from nearly the outset. Our best guess is that this could be caused by different visitors having drastically different speed machines and network connections. While there may be some way to normalize for this, we didn't have time to explore this further.

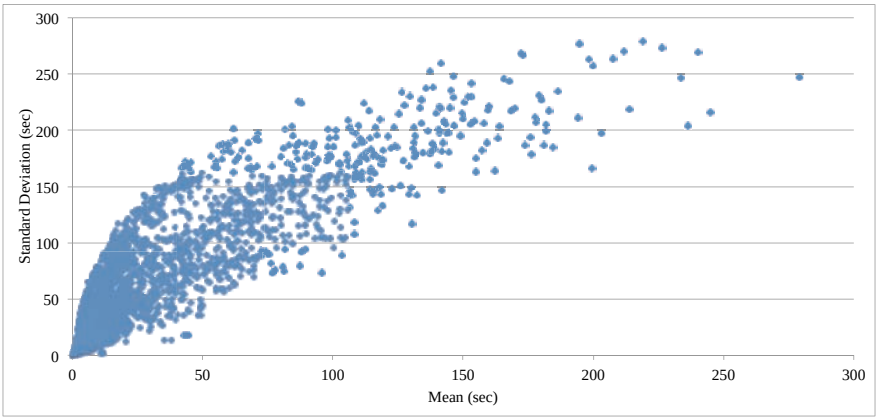

Figure 8: **Time between requested files**: We calculated the mean and standard deviation for all pairs of files where one was downloaded after the other by at least 20 distinct visitors. We had hoped there would be some kind of clear distinction between those that were automatically downloaded by the browser than by those that were manually downloaded. We saw no visible distinction.

Our third attempt was a simple stop-gap to solve some internal problems. Quite simply we classify files based on their file extension (and assume them to be HTML if it has no extension or an unknown extension). This is obviously incomplete, but we needed some way to limit the files we consider at various times (see Subsection 4.1 for HTML vs. not).

We believe there may be a useful solution somewhere in the intersection of these three attempts. It could be that by integrating the known weaknesses in each of the above, we could use each area's strength to compensate for another's weakness. We would also be thrilled with a more perfect fourth alternative!

7 Conclusion

We presented our results for a system to help analysts more quickly analyze web server logs. Our focus was on two fronts: First, we used statistical methods to group, filter, add data, and order visitors to hide less important and highlight the most suspicious visitors; second, we created novel interfaces and visualizations to give the user quickly understandable overview information for triaging.

We presented a statistical technique for separating browser-based visitors from crawler visitors based on the visitors' visit patterns. By leveraging visit patterns intrinsic to the browsers' and crawlers' purposes, we believe these metrics are fairly robust to avoidance. While it would be possible for crawlers to be re-written to act more browser-like on our metrics, it would likely decrease the crawler's effectiveness (as it would have to download many more non-HTML files). While a browser could be rewritten to act more crawler-like, it would decrease the users' experience: Eliminating bursts would result in painfully slow download times; decreasing non-HTML downloads would result in poorly laid-out, image-free webpages.

We presented a triaging interface that provides the user with the most interesting visitors as well as considerable summary data. We expect that a variant of this interface could provide single-user traige data as well.

Finally, we presented a TreeMap-based visualization of our webspaces, and showed how it can also be used to help analysts gain a quick "big picture" view of a visitor's (or pair of visitors') visits.

Each of these components were intended to help an analyst sift through hundreds of thousands of visitors per day. Our over-arching goal was to help analysts identify possible surveillance being performed in preparation for a spear phishing attack. We believe that if tools such as ours were used to sift through logs every day, analysts may be able to identify possible spear phishing targets and increase protections around them before a spear phishing attack succeeds. This would change security postures from a reactive response to attacks to a proactive attack defense and mitigation. Such a change could blunt the effectiveness of spear phishing – a critical step if we are to protect our computing infrastructure.

References

[Pre, 2013] (2013). Prefuse: Information visualization toolkit. http://prefuse.org/.

[Almeida et al., 1996] Almeida, V., Bestavros, A., Crovella, M., and de Oliveira, A. (1996). Characterizing reference locality in the web. In *Proceedings of PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems*, pages 92–107.

[Bruls et al., 1999] Bruls, M., Huizing, K., and van Wijk, J. (1999). Squarified treemaps. In *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization*, pages 33–42.

[Chen et al., 1996] Chen, M. S., Park, J. S., and Yu, P. S. (1996). Data mining for path traversal patterns in a web environment. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 385–392.

[Cialdini, 1993] Cialdini, R. B. (1993). *Influence: The Psychology of Persuasion*. Collins, revised edition.

[Cooley et al., 1997] Cooley, R., Mobasher, B., and Srivastava, J. (1997). Web mining: information and pattern discovery on the world wide web. In *Proceedings of Tools with Artificial Intelligence*, pages 558–567.

[Eirinaki and Vazirgiannis, 2003] Eirinaki, M. and Vazirgiannis, M. (2003). Web mining for web personalization. *ACM Transactions on Internet Technology*, 3(1):1–27.

[Iyengar et al., 1999] Iyengar, A. K., Squillante, M. S., and Zhang, L. (1999). Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100.

[McCullough, 2012] McCullough, D. (2012). White house confirms 'spearphishing' intrusion. http://news.cnet.com/8301-1009_3-57523621-83/white-house-confirms-spearphishing-intrusion/.

[Rivner, 2011] Rivner, U. (2011). Anatomy of an attack. http://blogs.rsa.com/rivner/anatomy-of-an-attack/.

[Shneiderman, 1992] Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transaction on Graphics*, 11(1):92–99.

[Shneiderman, 2013] Shneiderman, B. (2013). Treemaps for space-constrained visualization of heirarchies. http://www.cs.umd.edu/hcil/treemap-history/index.shtml.

[Srivastava et al., 2000] Srivastava, J., Cooley, R., Deshpande, M., and Tan, P.-N. (2000). Web usage mining: discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23.

[Wikipedia, 2012a] Wikipedia (2012a). List of HTTP status codes. http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

[Wikipedia, 2012b] Wikipedia (2012b). User agent. http://en.wikipedia.org/wiki/User_agent#Format.

[Wikipedia, 2012c] Wikipedia (2012c). X-forwarded for. http://en.wikipedia.org/wiki/X-Forwarded-For.

DISTRIBUTION

1 MS-0899 Technical Library 9536 (electronic copy)

Sandia National Laboratories