Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications

Peter Chapman
University of Virginia
pchapman@cs.virginia.edu

David Evans
University of Virginia
evans@virginia.edu

ABSTRACT

Web applications divide their state between the client and the server. The frequent and highly dynamic client-server communication that is characteristic of modern web applications leaves them vulnerable to side-channel leaks, even over encrypted connections. We describe a black-box tool for detecting and quantifying the severity of side-channel vulnerabilities by analyzing network traffic over repeated crawls of a web application. By viewing the adversary as a multi-dimensional classifier, we develop a methodology to more thoroughly measure the distinguishably of network traffic for a variety of classification metrics. We evaluate our detection system on several deployed web applications, accounting for proposed client and server-side defenses. Our results illustrate the limitations of entropy measurements used in previous work and show how our new metric based on the Fisher criterion can be used to more robustly reveal side-channels in web applications.

1. INTRODUCTION

Communication between the client and server in a web application is necessary for meaningful and efficient operation, but without care, can leak substantial information through a variety of side-channels. Previous work has demonstrated the ability to profile transfer size distributions over encrypted connections in order to identify visited websites [6, 8, 30]. Today, such side-channel leaks are especially pervasive and difficult to mitigate due to modern web development techniques that require increased client-server communication [5]. The competitive marketplace encourages a dynamic and responsive browsing experience. Using AJAX and similar technologies, information is brought to the user on demand, limiting unnecessary traffic, decreasing latency, and increasing responsiveness. By design, this approach separates traffic into a series of small requests that are specific to the actions of the user [27].

Analyzing network activity generated by web applications can reveal a surprising amount of information. Most attacks examine the size distributions of transmitted data, since the most commonly used encryption mechanisms on the web make no effort to conceal the size of the payload. As a result, traffic patterns can be correlated with specific keypresses and mouse clicks. Fundamentally, these attacks leverage correlations between the network traffic and the collective state of the web application.

To assist developers who want to create web applications that are responsive but have limited side-channel leaks, we developed a system to automatically detect and quantify the side-channel leaks in a web application. Our system identifies side-channel vulnerabilities by extensively crawling a target application to find network traffic that is predictably associated with changes in the application state. We use a black-box approach for compatibility and accuracy. Driving an actual web browser enables the deployment of our tools on any website, regardless of back-end implementation or complexity. Further, by generating the same traffic as would be seen by an attacker we ensure that information leaks due to unpredictable elements such as plug-ins or third-party scripts are still detected.

Previous work has used the concept of an attacker's ambiguity set, measured either in reduction power [5, 22, 33] or conditional entropy [21, 33] to measure information leaks. Our results show that entropy-based metrics are very fragile and do not adequately measure the risk of information leaking for complex web applications under realistic conditions. As an alternative metric, we adopt the Fisher criterion [11] to measure the classifiability of web application network traffic, and by extension, information leakage.

Contributions and Overview. We consider two threat models for studying web application side-channel leaks: one where the attacker listens to encrypted wireless traffic and another where an attacker intercepts encrypted network traffic at the level of an Internet service provider (Section 3.2).

We present a black-box web application crawling system that logs network traffic while interacting with a website in the same manner as an actual user using a standard web browser, outputting a set of web application states and user actions with generated network traffic traces, conceptually represented as a finite-state machine (Section 3). We have developed a rich XML specification format to configure the crawler to interact effectively with many websites (Section 4.2).

Using the finite-state machine output of the web application exploration, we consider information leaks from the perspective of a multi-class classification problem (Section 5). In building our example nearest-centroid classifier, we enumerate three distance metrics that measure the similarity of two network traces. Using the same set of distance metrics, we measure the entropy of user actions in the web application in the same manner as prior work and show that the variation and noise in real web applications make the concept of an uncertainty set insufficient for describing information leaks (Section 5.2). This motivates an alternative measurement based on the Fisher criterion to quantify the classifiability and therefore information leakage of network traces in a web application based on the same set of distance metrics (Section 5.3).

CCS 2011, October 17–21, 2011, Chicago, Illinois, USA.

We evaluate our crawling system and leak quantification techniques on several complex web applications. Compared to entropy-based metrics, we find the Fisher criterion is more resilient to outliers in trace data and provides a more useful measure of an applications vulnerability to side-channel classification (Section 6).

2. RELATED WORK

The study of side-channel vulnerabilities extends at least to the World War II [13]. Pervious work has considered side-channel vulnerabilities in a wide variety of domains, including cryptographic implementations [19], sound from dot matrix printers [1], and of course, web traffic [5, 6, 8, 30]. The most effective side-channel attacks on web applications have examined the size distributions of traffic [6, 8, 30]. The vulnerabilities stem from the individual transfer of objects in a web page, which vary significantly in size and quantity. Furthermore, due to the deployment of stream ciphers on the Internet [5], the sizes of objects are often visible in encrypted traffic. Tunneled connections [3, 21] and even encrypted wireless connections [3, 5] are vulnerable to these attacks.

The interactive traffic of modern web applications presents attackers with rich opportunities for side-channel attacks. Chen et al. demonstrated how an attacker could identify specific user actions within a web application based on intercepted encrypted traffic [5]. The leaks they found in search engines, a personal health information management system, a tax application, and a banking site required application-specific mitigation techniques for adequate and efficient protection. Search engine suggestions are a suitable example to demonstrate these attacks. As the user types a search query, the client sends the server the typed keys and the server returns a list of suggestions. An attacker can leverage the fact that the size of the suggestions sent after each keystroke vary depending on the typed letter to reconstruct search terms. Figure 1 shows how a single query is divided into a revealing series of network transfers. For a single letter, the attacker only needs to match the traffic to a set of 26 possibilities (letters A through Z). With the next letter, the attacker can use the reduced set of possibilities given by the first letter to drastically reduce the search space.

Side-Channel Vulnerability Detectors. Zhang et al. created Sidebuster [33], a tool that automates the detection of side-channel vulnerabilities in web applications built with the Google Web Toolkit (GWT) [16]. GWT developers write both the client and server code in Java and the framework generates the JavaScript code for the browser and automatically manages asynchronous communication. Every instance in the code where the client and server interact can

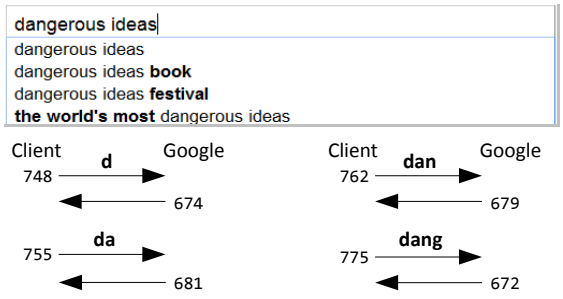


Figure 1: Search engines leak queries through the network traffic generated by auto-complete suggestions. The numbers indicate the number of bytes transferred.

be found through a straightforward static analysis. Sidebuster then quantifies the leaks using *rerun* testing, which initiates the statically discovered communication-inducing action using a simulated browser called HtmlUnit [4] and records the network traffic with Jpcap [14]. It measures the leaks by calculating the conditional entropy for each tested action. Sidebuster was tested on several demonstration applications and mock websites and shown to discover significant information leaks. We choose a black-box approach, in part, to create a leak discovery tool that is not limited to a particular implementation platform.

Black-box Web Application Exploration and Testing. Black-box exploration of traditional applications is often performed as part of test case generation [18, 23]. Many commercial automated black-box application security analysis tools are available (but none yet consider side-channel leaks) [2]. In the context of modern websites, black-box exploration is difficult since technologies such as AJAX break the traditional concept of a page. Crawljax was developed specifically to address the need to crawl the growing number of web sites employing AJAX elements [24, 25]. Our tool is built by extending Crawljax (see Section 4).

Side-Channel Leak Quantification. Current practice for quantifying the severity of web side-channels involves measuring the size of the attacker's uncertainty (or ambiguity) set in terms of reduction power [5, 22, 33] or bits of entropy [5, 8, 21, 33] as established in work measuring information flow in traditional software systems [26] and work specific to the web [20]. A primary goal is to quantify on average how well an attacker can determine the private state of the web application given a network trace. In that case it is also typical to simply measure the performance of a constructed classifier [6,22,30]. Section 5 discusses building a network trace classifier, measuring leaks in entropy bits, and our proposed method for quantifying leaks with the Fisher criterion.

Proposed Defenses. Prior work has developed a wide range of mitigation techniques for web side-channel leaks including random packet padding [5, 6, 8, 22], constant packet size [5, 6], and additional background traffic [6, 22]. The different defense strategies can be implemented in various points in the client-server interaction: in the application or server level of the host [5], through a local proxy [22], or in the web browser itself. The information available to the attacker affects the analysis and defenses of these leaks. While early work focused on intercepted HTTPS traffic [6], other work has considered other scenarios such as eavesdropping over a WPA/WPA2 connection [3, 5, 22]. Luo et al. developed HTTPOS, a client-side defense for the leaks [22]. HTTPOS is a local proxy that obfuscates network traffic by manipulating a wide range of HTTP, TCP, and IP protocol options and features such as adjusting the TCP window size, initiating TCP retransmissions, introducing timing delays, and even creating fake content requests. They target four threat models, two of which are shared with our work. In examining the applicability and effectiveness of packet padding defenses, Chen, et al. found that applications required specific and customized mitigation techniques, and proposed a development process for discovering, applying, and tuning defenses. We do not attempt to develop new defenses here, but rather to enhance understanding of side-channel leaks and to offer a side-channel leak quantification tool that can be used as a part of a mitigation process for typical web applications.

3. OVERVIEW

We built a completely dynamic, black-box side-channel vulnerability detection system. Figure 2 shows an overview of the system

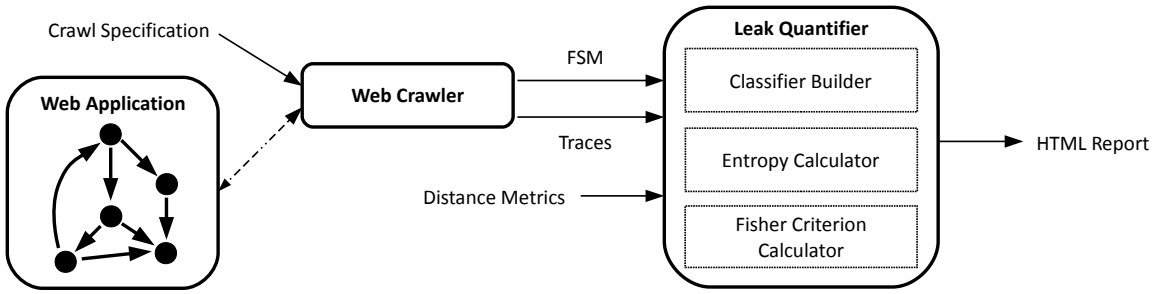


Figure 2: System Overview.

from the perspective of a developer. Using our system, a developer first creates an XML specification that assists the crawler in exploring the target website. The Web Crawler logs traffic while traversing the site (Section 4) and upon completion of a representative finite-state machine the Leak Quantifier analyzes the data to find relationships between user actions and network traffic (Section 5). A developer can use generated reports to pinpoint vulnerable areas of the site and devise effective and efficient mitigations. We do not attempt automatic mitigation here, although we believe the results produced by our tool could be used to automatically mitigate many leaks. Section 3.1 explains why we target a black-box solution; Section 3.2 clarifies the two threat models we consider.

3.1 Black-Box Analysis

Our approach does not assume any access to the server other than through standard web requests issued by a browser. There are many advantages to using a black-box, client-side only approach to perform the analysis. Generating actual user traffic makes experimental testing as close as possible to a real attack. Furthermore, a full browser such as Firefox can download and execute third-party scripts and plug-ins (e.g., Flash) which are crucial to realistic analysis since they could very easily be the the source of the leak. For example, an instructional Flash video could automatically play whenever the user of a tax site indicates they wish to obtain a certain tax credit. Although the remainder of the page may not result in distinguishable traffic, the streaming video could provide a clear indicator of the current state of the application.

Another advantage of a black-box approach is that our tool can be applied to any web application, regardless of its internal configuration. Section 6 reports on our experience applying our tool to several popular web applications. The primary limitation of the applicability of our system is that a standard Selenium installation cannot interact with items that are outside of the browser DOM (e.g., embedded Flash objects) or that do not fit well into the traditional web page model (e.g., HTML5 Canvas). Ongoing work is attempting to add stable Flash support to Selenium [12] which could be integrated into our testing framework.

3.2 Threat Model

We consider two threat models in this work: an attacker eavesdropping on encrypted wireless traffic and an attacker scanning through traffic directed through an ISP. For both cases, we assume an attacker targets a particular individual to learn as much information as possible from their encrypted web browsing. Most of our results may also apply to the scenario where a government agency, for example, is scanning a large amount of traffic from unknown individuals to find evidence of particular transactions, but we do not consider that scenario in this work.

WiFi Snooping. In the WiFi Snooping threat model, an attacker collects data over an encrypted wireless network. Example targets include high profile persons such as a politician or CEO, about whom sensitive information could be valuable to competitors or abused in devious ways. For example, one corporation could eavesdrop on its competitor's CEO's search queries to anticipate the competitors entry into a new market. Another possible attack would be a con-artist exploiting leaked sensitive information to customize a scam a particular victim. In our model, the WiFi snooper can see the size of network transfers and whether they are incoming our outgoing from the client but no other information about the data. We believe this model accurately reflects what an eavesdropper could learn since the access point (AP) announces its MAC address in the AP beacons and there are a variety of ways for the attacker to infer the target's MAC address.

ISP Sniffing. In our second threat model, the adversary taps directly into the traffic flowing through an ISP either with legal authority or by compromising network equipment. Such an attacker can observe the IP and TCP plain text packet headers in HTTPS communication including the source IP, destination IP, and the size of the encrypted payload.

4. CRAWLING WEB APPLICATIONS

The Crawler explores the target web application to build a finite-state machine (FSM) representation of the site. Each state in the FSM is a possible state of the DOM (Document Object Model) in the application saved as an HTML file. The transitions between states are the user actions that load new pages or trigger DOM changes, annotated with recorded network traffic over repeated trials. The FSM is the input into our leak quantifier which measures the degree to which paths through the state machine are consistent and identifiable.

Figure 3 shows the structure of the Crawler. We extend the Crawljax tool to manage the crawling process (Section 4.1). Selenium automates the Firefox actions needed to interact with the target web application. We use Jpcap to collect a network trace. Since an exhaustive crawl is not possible for any interesting web application, we also extend Crawljax to allow developers to specify a directed crawl using a Crawl Specification, as described in Section 4.2. Since web services are often unreliable, it is necessary to repair the resulting FSMs, as described in Section 4.3.

4.1 Crawljax

Crawling modern web sites and services is challenging because of their highly dynamic nature and emphasis on client-side tech-

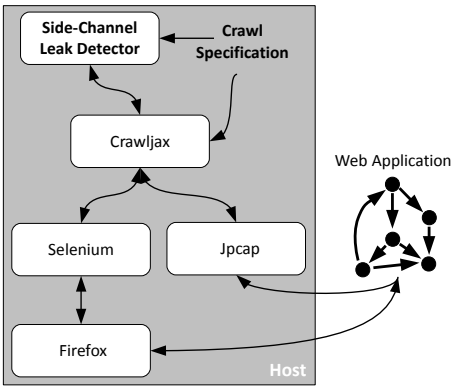


Figure 3: Crawling Web Applications.

```
<specification>
 <click>
  <tag>a</tag>
    <xpath>/HTML/BODY/DIV[2]/DIV[2]/A</xpath>
 </click>
 <waitFor>
   <id></id><value>footer</value><url>.</url>
 </waitFor>
 <matchOverride>
  <match>
   <pattern>
    <![CDATA[[\w\W]*(name="currentquestion")[\w\W]*?(<\/fieldset>)]]>
   </pattern>
   <replace></replace>
  </match>
 </matchOverride>
</specification>
```

Figure 4: Example Interaction Specification.

nologies that violate the traditional concept of a web page [25]. We build upon Crawljax [24], an open-source tool designed to check that sites are searchable, testable, and accessible. Crawljax attempts to construct a state machine of user interface states for web applications even in the presence of JavaScript and AJAX [25]. A web application state, with the default Crawljax settings, is a specific DOM configuration. If user actions such as clicks or keyboard actions result in changes to the DOM, Crawljax creates a new state and connects the two with a transition. To accomplish this, Crawljax drives an instance of the Selenium [15, 29] testing framework for black-box manipulation and state inference of the application. To support our goal of black-box detection of side-channel vulnerabilities of complex web applications, we made several changes to Crawljax, described next.

Logging Network Traffic. During execution Crawljax interacts with developer-specified elements and forms while monitoring the browser DOM to construct a corresponding state machine. We added network traffic logging to Crawljax. We used the existing plugin architecture built into Crawljax and the Jpcap Java library for network packet monitoring [14].For our experiments, we logged packet source, destination, length, and inter-packet timings. To improve robustness, our network plugins are also aware of basic TCP features such as sequence numbering and retransmission. Importantly, our tools do not use any knowledge unavailable to a potential adversary intercepting SSL-encrypted traffic. Experiments in the WiFi threat model ignore TCP features.

Caching. Unlike most previous experiments, the browser cache was left enabled to more accurately model typical web traffic. With Crawljax's depth-first search methodology, the cache is reset upon returning to the root of the website, which retains the functionality an attacker would face as the user goes through the site. Since Selenium does not currently include such functionality [28] we wrote a Firefox extension that communicates with our system.

4.2 Crawl Specification

Ideally, we would exhaustively visit every state of a web application. For any non-trivial application, state explosion makes such a goal infeasible. Hence, we developed a crawling specification that can be used to direct the navigation. A developer may provide three XML specifications to our system described in the following subsections: a required interaction specification that directs the crawl, an input specification for handling input fields, and a login specification for managing accounts. Regarding the developer burden of using our tool, writing the specification files is not harder than designing tests cases with the Selenium framework, a widely used tool for black-box web application testing. The primary task for the user of our system is to identify sections of the site that contain private interactions using the XPath notation and private content on the page with regular expressions and we advocate the use of of our tool as part of the development of privacy-sensitive applications.

Interaction Specification. At a minimum, a crawl specification must specify which elements on a site to click. This specification can specify click elements by tag, attribute, or an XPath [7]. Figure 4 shows an example of a interaction specification for the NHS Symptom Checker (see Section 6.3). It instructs the crawler to click on all anchor tags that satisfy the XPath expression, /HTML/BODY/DIV[2]/DIV[2]/A. This corresponds to the "next" button in the questionnaire. During exploration, the crawler will scan each page for elements satisfying the criteria, adding them to the depth-first search stack. The click element can be further refined with noClick elements that override the click specification.

A developer can use the waitFor element to specify a DOM element that indicates a web page has successfully loaded. This was added after we observed that occasionally a request will only be partially answered, causing only a portion of the page to load. When the described element appears in the DOM, Crawljax continues normally. If that particular request never completes fully, our system times out and retries the user action. The specification in Figure 4 tells the crawler to wait for the page footer to load.

By default, Crawljax defines two equivalent states as having a near identical Document Object Model representations (DOM). In reality, two instances of a web page may be semantically identical but have slightly differing DOMs. A developer can use matchOverride to enumerate a series of regular expression replacements to manipulate the DOM before the standard state comparison is performed. The example matchOveride in Figure 4 directs the crawler to only examine the questions from the symptom checker for the state equivalence computation.

Input Specification. Many websites require user input for meaningful use. Due to the difficulties of inferring valid inputs, the developer must specify where and how to fill input fields in the application. The example input specification shown in Figure 5 was used to select the gender in NHS Symptom Checker. An input specification indicates how the fields will be populated with the beforeClickElement, which in this case would be the next button in the gender form. Additionally, the developer lists which field should

```
<form>
 <beforeClickElement>
   <tag>input</tag>
   <xPath><expression>
     /html/body/span/center/span/center/form[@id="gender"]/table/tbody/tr/td
          [2]/div/input
   </expression></xPath>
 </beforeClickElement>
 <field>
  <id>female</id><value>false</value><value>true</value>
 </field>
 <field>
  <id>male</id><value>true</value><value>false</value>
 </field>
</form>
```

Figure 5: Example Input Specification.

be populated and all the possible inputs to try. The field tags in the example direct the crawler to first select the female option, follow that line of questions, and then return to select male. Specifications can also request performing a random subset of specified inputs. For example a developer may want to try random combinations of first and last names in order to get a larger sample. It may also be the case that progression in the application requires valid input (e.g., a Social Security Number), but the developer does not want network traffic to be tied to a specific input. Such functionality is implemented with the randomForm element.

Login Specification. Many real-world applications require existing user-accounts to function. Google Health requires login credentials to access the site. We extended Crawljax's basic support for form pages to allow the developer to list a series of accounts, from which one is chosen for a particular crawl. Using different accounts prevents logged traffic information from being over-fitted to a specific user. An example login specification for Google Health is shown in Figure 6. In this specification a URL is given to login at, the username and password along with where to input them, and what button to click to complete the login.

4.3 Crawl Repair

Since our crawling system triggers thousands of page loads when exploring externally operated complex web applications, errors often occur in the crawl. The most common failures were unfulfilled HTTP requests and generic application-level server-side errors. These failures result in incorrect network traces and even structurally different state machines, since application error pages or incomplete page loads prevent the crawler from finding new DOM elements with which to interact.

After a series of crawls the developer selects a trial as the ground-truth, presumably through manual analysis of the saved HTML states. We developed an accompanying tool to examine the suspect trials and repair the FSMs so that they are structurally equivalent. This involves removing states and transitions that do not exist in the selected trial and adding those as necessary. When adding new states to imperfect crawls we choose to disadvantage the attacker by assuming that no information was gained from that particular state transition in that trial. If desired, one could give the advantage to the attacker by replacing the missing data with the trace from the designated correct trial. In practice we found that within a trial the number of discovered errors is small; fewer than 1% of the number of total transitions need corrections.

```
<preCrawl>
 <url><value>http://www.google.com/health</value></url>
 <input>
   <id>Email</id> <value>test@xkcd.com</value>
 </input>
 <input>
   <id>Passwd</id> <value>Tr0ub4dor&3</value>
 </input>
 <click><id>signIn</id></click>
</preCrawl>
```

Figure 6: Example Login Specification.

5. LEAK QUANTIFICATION

Once the site exploration phase is complete, the leak quantifier analyzes the state machine of the web application to determine how vulnerable the network traffic is to reconstruction through the various side-channels. Each state transition contains a list of network transfers with information about the origin, destination, size, and time of the transfer as described in Section 4.1. To determine the similarity of two traces, we define a *distance metric*. Section 5.1 describes three different distance metrics we use based on different aspects of the network traces. Then, we consider two methods to quantify leaks in the web application: entropy (Section 5.2) and the Fisher criterion (Section 5.3).

Assumptions. A key assumption made throughout our leak quantification is that the adversary is able to track when state transitions begin and end. This is reasonable since the adversary can search for pauses in network traffic. For most web applications it is impractical to continually stream data between the client and the server due to the computational overhead and the bandwidth consumption, so traffic bursts reflect state transitions. For the WiFi threat model, we assume there is no other disruptive network traffic and that the attacker can distinguish whether packets are incoming to or outgoing from the victim, which is essentially a matter of identifying the MAC address of the target computer. Our last assumption is that the user starts at the root of the web application (e.g., the Google Health Dashboard or the NHS Symptom Checker welcome page) and makes forward progress through the application, not clicking back or randomly reloading pages. These assumptions favor the attacker, so potentially overestimate the amount of information available to an attacker in practice, although it seems likely that motivated attackers would be able to find ways to overcome violations of these assumptions.

Formalization. To formalize the problem as a multi-class classifier, we define x_i to be the set of examples belonging to class i, and x_i^j to be example j from the class. A *class* is the action or series of actions a user performed to reach a state in the web application. An example from a class is the set of network traces collected while those actions were performed. We assume the start and stop of page transitions are identifiable, so the function $t(x_i^j, k)$ yields the trace from example x_i^j for the k^{th} transition. A network transfer is defined as the uninterrupted transmission of data (generally over TCP for our purposes) from one machine on the network to another. A trace is a list of network transfers of the form $src \rightarrow dst : bytes$ where src is the source, dst is the destination, and $bytes$ is the number of bytes of the transfered data as taken from the IP header. Given a transition v, $v[i]$ yields the i^{th} transfer.

Impact of Threat Models. The threat models dictate the amount of information visible to the attacker. For the WiFi scenario, the attacker can only see the size of transfers and whether they were

incoming or outgoing. Thus, all transfers are of the form $target \rightarrow accesspoint$ or $accesspoint \rightarrow target$. In the ISP threat model, the attacker has access to the plain text IP and TCP packet headers in in addition to the encrypted contents of the message. Since the ISP scenario allows the attacker to see the TCP packet headers, TCP protocol features such as ACKs and re-transmissions are easily identified. In anticipation of defenses designed to abuse these protocol features by sending fake ACKs or initiating unnecessary re-transmissions (as suggested by HTTPOS [22]), tests under the ISP threat model ignore both ACKs and re-transmissions. Unlike our other assumptions, this assumption favors the site operator by assuming the attacker's task is complicated by widespread deployment of these defenses, so underestimates the actual leakage in situations where these defenses are not used.

Baseline Classifier. To use as a baseline for testing the existence and exploitability of side-channel leaks, we construct a multi-class classifier, classifying network traces according to the action performed to generate them. Our classifier uses a nearest-centroid approach, assigning an unknown trace to the class of the nearest class centroid, where nearest is defined according to one of the distance metrics. Since the exact distribution of each class is unknown we estimate the centroid by attempting to create a trace that minimizes the Hamiltonian distance from the examples in the class. We validate the performance of the classifier by running K-fold cross-validation testing. The higher the success rate of the classifier, the more likely an attacker will be able to exploit a leak based on the properties measured in the metric. Ideally, a well protected system would not allow an attacker create a classifier that performs better than is possible with random guessing.

5.1 Distance Metrics

We use different distance metrics to test different environmental conditions and threat models to understand how conditions impact what vulnerabilities exist and the best methods to mitigate them.

Total-Source-Destination. The Total-Source-Destination (TSD) metric returns the summed difference of bytes transferred between each party. In a trace containing only a server and a client, it is the difference in the number of bytes transfered to the client, added to the difference in the number of bytes transfered to the server, as computed by Algorithm 1. The inputs are two transitions v and w, and the output measures the distance between the transitions. This metric is easily manipulated through basic packet padding which hides the actual lengths of the packets.

Algorithm 1 *TotalSourceDestination*(v,w)

```
distance = 0
for all s ∈ Parties do
  for all d ∈ Parties do
    subdistance = 0
    for all i = 0 → v.size do
      if v[i].src = s ∧ v[i].dst = d then
        subdistance = subdistance + v[i].bytes
    for all i = 0 → w.size do
      if w[i].src = s ∧ w[i].dst = d then
        subdistance = subdistance − w[i].bytes
    distance = distance + abs(subdistance)
```

Size-Weighted-Edit-Distance. The Size-Weighted-Edit-Distance (SWED) adds robustness by tracking the control-flow of the transfered information. Unlike the Total-Source-Destination metric, the sequence of transfered data matters. Every transfer is treated as a symbol in a string of the sequence of transfers, $src \rightarrow dst$. The distance is the Levenshtein distance between the translated strings, but in order to give weight to the transfer sizes, the cost of each edit operation (insertion, deletion, and substitution) is the number of bytes being inserted or deleted. A minimum weight is set at a configuration value α, in order to lend sufficient weight to smaller transfers (TCP ACKs). If the source and destination are the same, the cost is simply the difference in transfer sizes.

Edit-Distance. Since the simple packet-padding defense dramatically affects the size distributions of transfers, we use the Edit-Distance (ED) metric to understand how well an attacker can do using only on the control flow of the network transfer. Like the previous metric, every transfer in the trace is a symbol in a string. The Edit-Distance is the Levenshtein distance between two strings where all edit operations have an equal cost. Since this metric is independent of the sizes of transfers, the Edit-Distance reveals how well an attacker can do against a perfect packet-padding strategy.

Random. The Random metric serves as a baseline in order to judge the distinguishability gained from the distance metrics beyond the assumption that the adversary can distinguish page breaks. In every metric, the nearest-centroid classifier will not consider classes that require a different number of transitions than the example in question. The Random assigns a random distance between 1 and 1000 regardless of the two examples being compared. Hence, the only useful classifiability gained from the Random metric is a result of the assumption that the adversary can identify when state transitions occur.

5.2 Entropy Measurements

Previous work measured the severity of leaks using bits of entropy [21, 33] or reduction power [5, 22, 33]. Both measurements are a function of the size of the *uncertainty set* the attacker has for classifying a given network trace. In other words, given a network trace, how many classes can be eliminated as an impossibility for generating that trace. Logically, bits of uncertainty indicate the amount of ambiguity an attacker has in classifying a network trace. Using a concrete example, if a network trace is identical for four actions that trace is said to have $\log_2 4 = 2$ bits of entropy. Ideally we would measure the entropy for every possible network trace, looking at the number of classes that could possibly create each trace. To find the entropy of the system, we sum the entropy of each trace weighted by the probability of that trace occurring. In practice, however, it is infeasible to enumerate every possible trace so we use the corpus of those generated by our testing. To simplify our model we assume that each user action is equally probable. Together, the equation for calculating entropy is:

$$H(X) = \sum_{i=0}^{n} \frac{\log_2 p(\bar{x}_i)}{n}$$

where X is the tested system containing n classes, $\bar{x}_i$ is each centroid, and $p(\bar{x}_i)$ yields the size of the uncertainty set for the attacker. Note that if the uncertainty set is n for every trace, the resulting entropy is maximized at the desired $\log_2 n$. This is the conditional entropy metric used by Luo et al. [22].

The key difficulty in calculating entropy lies in determining the size of the uncertainty set for a given trace. In our analysis we take the estimated centroid for each class, then find the threshold distance from the centroid such that a certain percentage of the samples in the class are within that distance of the centroid. We use the threshold distance as the boundary for distinguishability. The number of centroids that fall within the threshold distance of the centroid yields $p(x_i)$ for this class. Figure 7 shows two classes con-

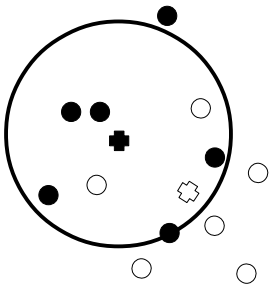


Figure 7: Entropy Distinguishability Threshold. Two classes are marked by different shadings with their respective centroids indicated by the + symbols. The 75% threshold for the dark class is the distance that contains $3/4$ of the dark points. Since the centroid of the light class is within this threshold, we consider the classes indistinguishable at this threshold.

sidered indistinguishable given a threshold of of 75%.

The threshold for distinguishability in calculating entropy is often arbitrary. Depending on the choice, the resulting entropy value may not give a good measure of the attacker's likelihood of successfully exploiting a particular vulnerability (as demonstrated by our experimental results in Section 6). Additionally the boundary between distinguishable and indistinguishable classes is not necessarily strict and small changes can yield significant changes in entropy. It is desirable for our leak quantification to capture the relative distances of classes. Ignoring the relative distances can lead to misclassifying a system as invulnerable because each class appears indistinguishable but slight changes to the attacker strategy may yield an accurate classifier.

Ideally, defenses should generate network traces that are either exactly the same or entirely randomly distributed. If under the monitored properties the traces are either identical or entirely randomly distributed, the data is invulnerable to side-channel analysis. Our measurement should yield a meaningful value for this result. However, once the entropy measurement reaches its maximum value, each class is considered indistinguishable from every other class ignoring any notion of how distant or different classes are from one another. Two defenses that have maximum entropy values are not necessarily equally good considering a defense that barely establishes indistinguishability will be given the same value as a perfect defense. Our second metric is designed to overcome these limitations of the entropy measurement.

5.3 Fisher Criterion

Since we frame the goal of the attacker as a classifier, it is natural to borrow concepts from machine learning methods in constructing classifiers. We adopt the Fisher criterion as our measurement of classifiability [11]. The Fisher criterion was previously used by Guo et al. as the fitness function for a genetic programming algorithm to extract meaningful features for multi-class recognition problems [17], but we are not aware of any previous use in side-channel analysis.

The Fisher criterion is essentially the ratio of the between-class variance to the within-class variance of the data [31,32]. The higher the value, the more classifiable the data. The Fisher criterion is used as a tool in linear discriminant analysis to construct strong classifications. Since we are given the classifications (it is known which user actions created the network traces), we use the Fisher Criterion as a measurement of the severity of side-channel leaks. The Fisher criterion is calculated as:

$$F(X) = \frac{\sigma^2_{between}}{\sigma^2_{within}} = \frac{\sum_{i=0}^{n} m \cdot (\bar{x_i} - \bar{x})^2}{\sum_{i=0}^{n}\sum_{j=0}^{m} (x_i^j - \bar{x_i})^2}$$

where n is the number of classes, m is the number of samples in each class, x_i^j denotes sample j in class i, $\bar{x_i}$ is the centroid of class i, and $\bar{x}$ is the total centroid. A Fisher criterion value greater than 1 has the physical meaning that the between-class variance is greater than the within-class variance. Although this may seem like a logical threshold for distinguishable classes, as has been previously claimed [31], our results do not support the existence of an absolute threshold.

The Fisher criterion is a better measurement of the classifiability of network traffic than previous entropy measures for two reasons: (1) it incorporates the distances between classes without the almost arbitrary distinction of distinguishable versus indistinguishable, increasing robustness against attack variations; (2) both the ideal and worst-case network trace distributions have associated values, 0 and ∞ respectively. The Fisher criterion approaches zero because either the within-class variance approaches infinity (the values within a class are random), or the between-class variance approaches 0 (all classes yield the same network traces). The Fisher criterion approaches infinity when the classes are well-separated and are well-defined, lending well to strong classifiability.

6. RESULTS

To evaluate the effectiveness of our black-box approach in side-channel leak quantification and the value of the Fisher criterion over conditional entropy, we tested our system on several existing web applications: search engines (Section 6.1), Google Health (Section 6.2), and the United Kingdom's National Health Service Symptom Checker (Section 6.3). Some of the search engines were tested and the Google Health application were also used in prior work [5]; the NHS Symptom Checker was chosen because it is a complex application that handles sensitive information.

For each application, we constructed crawl specification files as described in Section 4.2 and ran the crawlers on a variety of commodity hardware including desktops, laptops, and servers. The difficultly of writing these specifications varies as a function of website's complexity and adherence to standard web design practices such as using a RESTful [10] architecture and avoiding iframes. The average length of the constructed specifications is 4547 LOC ($\sigma = 7537$) according to CLOC [9]. A detailed breakdown of specification sizes can be found in Table 1. The Yahoo Search specification was the longest (17,589 LOC) as it includes an (automatically-generated) enumeration of three-letter combinations and the Bing specification was the shortest (45 LOC). Once the crawlers finish exploration of the web application, we quantified the leaks. The results of the leak quantification for each application are presented in the following subsections. During all tests, the browser cache was left enabled, but reset upon returning to the root of the web application to ensure that the elements in the cache are only a function of the pages visited from the root.

We developed our tool and helper extensions for Firefox 3.6, although they could be adopted to any browser supported by Selenium. In fact, comprehensive site analysis may require using multiple browsers since uneven support of web standards may significantly vary the traffic signature from one browser to another. We have used our system on a variety of different systems running Windows XP, Windows 7, Ubuntu 9.10, and Ubuntu 10.04. Crawling

Application	Interaction	Input	Login
Bing Suggestions	3	41	-
Google Search Suggestions	3	38	-
Google Instant	3	38	-
Yahoo Search	3	17589	-
Google Health	31	324	82
NHS Adult Male	37	286	-

Table 1: Specification Length. The 17589-line specification for Yahoo Search is automatically generated by enumerating all 3-letter combinations. The other specifications are manually generated. Only the Google Health specification includes a Login specification, since the other applications do not require user accounts.

a web application, just like performing any depth-first search, is trivially parallelized by assigning different instances to crawl different subtrees of the site. In addition to commodity desktops and laptops we tested our setup on a 64-machine cluster, demonstrating the ability for a developer to run very large crawls consisting of tens of thousands of pages in a matter of hours.

Section 6.4 uses our tools to analyze HTTPOS [22], a defense against side-channel attacks on the web. In Section 6.5, we test our results against a suite of general-purpose machine learning algorithms to confirm that our domain-specific methods are better than the best available general-purpose techniques.

6.1 Search Engine Suggestions

Chen et al. demonstrated how the Bing (http://bing.com), Google (http://encrypted.google.com), and Yahoo (http://search.yahoo.com) search engines leak queries through the network traffic generated by search suggestions [5]. Suggestion fields are particularly vulnerable to side-channel attacks because they update with every keystroke. Bing and Google search suggestions begin appearing after a single lowercase letter, so they were tested by scripting the typing of a single letter and measuring the accompanying network traffic. As demonstrated by Chen et al., the ability to distinguish a single letter allows the attacker to build up the entire query [5].

In September 2010, Google introduced Google Instant, which loads the search results as the user types a query. We evaluated Google Instant in the same manner as Bing and Google search suggestions. For Bing and Google search suggestions, classification performance is strong, reinforcing findings in prior work.

Yahoo's search suggestions do not begin appearing after three characters have been typed, increasing the state space for the first network transfer from 26 to $26^3 = 17,576$. We tested Yahoo Search to see how much delaying the suggestions mitigates the leak. The output of the classifier is a set of predicted classifications for a given example. The classification is considered correct if the actual example class is in the set.

Table 2 shows the results of our nearest-centroid classifier under the described metrics and proposed threat models for each search engine. Table 3 presents the entropy results. The distinguishability threshold greatly impacts the estimated bits of entropy in a query. For example, the classification accuracy using the Total-Source-Destination metric under the WiFi threat model is over 93% for Bing. The associated entropy calculation yields an average of 0.91 bits of uncertainty, meaning that on average the attacker's uncertainty set is $2^{0.91} = 1.88$. Considering the results of our classifier, 0.91 bits of entropy underestimates the classifiability of the data and a more appropriate 0.07 bits is only reached after ignoring the farthest 25% of sample points from the centroid. The inherent noise present in real world network traces shows the fragility of the entropy metric in the presence of classification outliers.

The calculated Fisher criterion values for the search suggestions in Table 4 give a more consistent view of the data while granting us new insights into the classifiability under the various metrics. Note that the Fisher criterion for the Edit-Distance metric is 0.00 for the search engine suggestions. This is logical considering the nature of search suggestion network traffic where almost every network trace is a short interaction between the client and the server consisting of a request, a response, and an acknowledgment. Under the Edit-Distance metric, each example trace is nearly identical and so reaches the goal Fisher criterion value of zero.

6.2 Google Health

We also tested our system on Google Health's (https://health.google.com) "Find a Doctor" feature. The "Find a Doctor" tool has been shown to leak the type of doctor a user searches and by extension a user's medical condition [5]. Since Google Health requires an account to function, we used the login functionality of our crawler described in Section 4.2. Using the application, the user inputs an area of medicine and a location. The crawler enumerates the areas of medicine in a drop-down menu to trigger searches for specialty doctors. The result of the search is a list of nearby doctors specializing in the requested medical field. As in Chen et al.'s work [5], we assume the adversary has a way to accurately determine the location (which is a reasonable assumption in cases where the adversary either knows the target's physical location or has access to the target's IP address).

The classifier performance (included in Table 4) is over 88% on the Google Health tool using the Total-Source-Destination metric, with similar results using Size-Weighted-Edit-Distance. The Edit-Distance metric yields little classification value, since like the search suggestions, the control-flows are largely similar. As seen in the other web applications, the entropy values (Table 3) decrease drastically as the threshold is decreased. However, we can observe that simply decreasing the threshold does not guarantee a representative result. For example, lowering the threshold to 50% with the Total-Source-Destination metric decreases the entropy to almost zero. Our classifier, on the other hand, is not able to classify roughly ten percent of the examples. Lowering the threshold in hopes of getting more accurate entropy values ignores actual sample points, even if they are outliers, and can result in underestimating the entropy. As expected from the classifier performance results, the Fisher criterion for the Edit-Distance under the ISP threat model is 0.00. Like the search suggestions, this is because the control flows are nearly identical for each query.

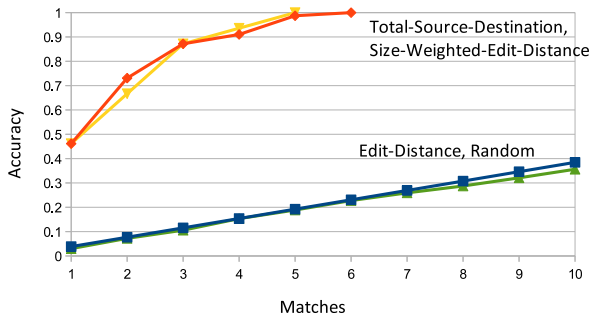


Figure 8: Performance of our Classifier. Our classifier performs well on Google Search suggestions when using the size-based metrics, but almost no better than random when sized is ignored.

Distance Metric		Bing		Google Search		Google Instant		Yahoo Search		Google Health		NHS	
Matches		**1**	**10**	**1**	**10**	**1**	**10**	**1**	**10**	**1**	**10**	**1**	**10**
Random		2.9	35.6	2.9	35.6	2.9	35.6	0.0	0.0	1.3	10.8	3.6	29.9
ISP	TSD	95.7	100.0	46.1	100.0	47.5	88.3	1.2	8.0	88.2	93.6	85.8	100.0
	SWED	96.3	100.0	46.1	100.0	7.3	52.6	1.1	7.9	81.8	91.9	31.0	89.7
	ED	3.7	37.0	3.8	39.5	7.7	56.0	0.0	0.0	2.0	11.1	5.8	38.3
WiFi	TSD	93.7	99.4	44.9	100.0	39.4	87.6	1.2	7.9	85.9	90.9	60.6	99.2
	SWED	94.7	98.8	44.9	100.0	29.6	83.0	1.2	7.9	81.8	89.6	46.9	97.7
	ED	3.7	37.0	3.8	38.5	31.5	86.7	0.0	0.1	2.7	19.9	46.1	98.1

Table 2: Nearest-Centroid Classifier Results. The value of *Matches* indicates the size of the set returned by the classifier. The results show the percentage of the time the correct classification is included time in the returned set of the given size. The metrics are Total-Source-Destination (TSD), Size-Weighted-Edit-Distance (SWED), and Edit-Distance (ED).

Distance Metric		Bing			Google Search			Google Instant		
Threshold		**100%**	**75%**	**50%**	**100%**	**75%**	**50%**	**100%**	**75%**	**50%**
Expected		4.70	4.70	4.70	4.70	4.70	4.70	4.70	4.70	4.70
ISP	TSD	0.42	0.07	0.07	0.42	0.07	0.07	4.70	1.97	1.09
	SWED	0.42	0.07	0.07	0.42	0.07	0.07	4.64	3.90	3.37
	ED	4.70	4.70	4.70	4.70	4.70	4.70	4.70	4.43	3.54
WiFi	TSD	0.91	0.07	0.07	2.95	2.40	0.44	4.70	2.02	1.02
	SWED	0.78	0.07	0.07	1.13	0.56	0.44	4.70	2.40	1.58
	ED	4.70	4.70	4.70	4.70	4.70	4.70	4.70	2.54	1.74
Distance Metric		**Yahoo Search**			**Google Health**			**NHS**		
Threshold		**100%**	**75%**	**50%**	**100%**	**75%**	**50%**	**100%**	**75%**	**50%**
Expected		14.01	14.01	14.01	6.63	6.63	6.63	8.87	8.87	8.87
ISP	TSD	7.86	6.80	5.05	0.58	0.05	0.01	5.13	2.83	1.92
	SWED	7.88	6.47	5.32	0.74	0.25	0.14	6.19	4.77	3.98
	ED	12.66	12.43	12.42	6.55	6.55	6.55	7.07	6.65	6.21
WiFi	TSD	7.91	6.62	5.04	0.71	0.05	0.01	4.76	2.70	1.83
	SWED	7.91	6.62	5.04	1.04	0.19	0.14	6.25	4.53	3.89
	ED	12.64	12.36	12.26	6.05	5.89	5.89	5.65	4.43	3.82

Table 3: Entropy Results (measured in bits of entropy)**.**

Distance Metric		Bing	Google Search	Google Instant	Yahoo Search	Google Health	NHS
ISP	TSD	5.18	4.13	1.13	0.69	12.1	4.9
	SWED	0.17	41.7	0.34	0.59	18.0	3.3
	ED	0.00	0.00	0.22	0.56	0.0	1.8
WiFi	TSD	6.04	4.13	0.84	0.59	11.3	5.4
	SWED	1.26	41.7	0.76	0.58	10.8	3.2
	ED	0.00	0.00	0.79	0.51	3.0	5.0

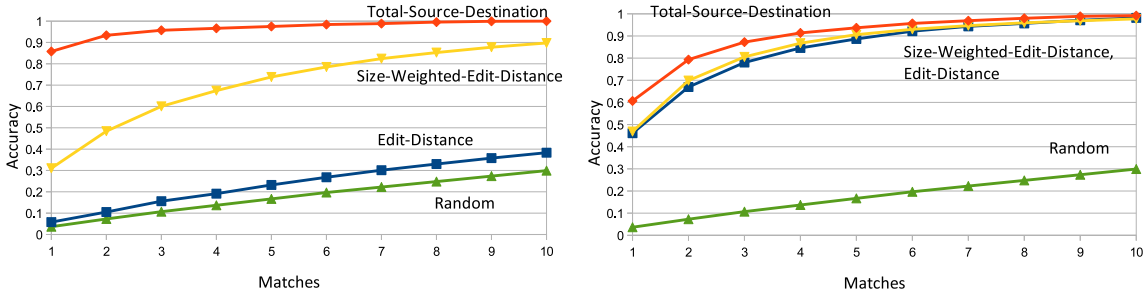
Table 4: Fisher Criterion Results.

Application		Google Search							Google Instant						
Distance Metric		**Accuracy** (%)			**Entropy** (bits)			**Fisher**	**Accuracy** (%)			**Entropy** (bits)			**Fisher**
		1	3	10	100%	75%	50%	**Criterion**	1	3	10	100%	75%	50%	**Criterion**
ISP	TSD	3.4	12.8	38.0	4.70	4.33	4.06	0.28	43.7	66.8	87.6	4.70	3.97	3.40	0.60
	SWED	3.8	11.1	38.0	4.70	4.43	3.52	0.43	8.2	20.4	51.4	4.16	3.61	3.55	0.55
	ED	3.4	9.4	35.5	4.70	4.58	3.51	0.14	8.7	19.0	55.0	4.70	4.55	3.81	0.47
WiFi	TSD	6.0	17.9	48.3	4.70	4.28	3.34	0.22	37.0	59.3	85.6	4.08	3.29	2.22	0.61
	SWED	3.8	11.1	35.0	4.67	4.46	3.91	0.23	27.2	47.6	82.2	4.38	3.70	2.67	0.57
	ED	6.8	11.1	35.5	4.70	4.52	3.93	0.37	26.2	49.8	81.5	4.16	3.61	3.55	0.69

Table 5: Leak Quantification Results for Google Search Suggestions and Google Instant while using HTTPOS.

Data Set	Best Classifier	Accuracy	Our Rate
Bing Suggestions	minimalist-boost	91.2	96.3
Google Search Suggestions	LogitBoost_weka_nominal	34.8	46.1
Google Instant	bonzaiboost-n200-d2	66.0	47.5
Google Health Find A Doctor	LogitBoost_weka_nominal	74.2	88.2
NHS Adult Male	FilteredClassifier_weka_nominal	78.1	85.8
HTTPOS on Google Search	bonzaiboost-n200-d2	7.1	6.8
HTTPOS on Google Instant	bonzaiboost-n200-d2	15.6	43.7

Table 6: MLComp Results. Running our datasets on generic, publicly available multi-class classifiers yields similar results to our nearest-centroid classifier. Each row of the table lists the best accuracy rate that any classifier had for that dataset as a percentage.



(a) ISP Threat Model

(b) WiFi Threat Model

Figure 9: Classifier Performance on NHS Symptom Checker.

6.3 NHS Symptom Checker

To analyze our metrics on a more complex privacy-sensitive site, we also conducted an experiment using the Symptom Checker created by the United Kingdom's National Health Service (NHS). The NHS symptom checker asks a visitor a series of multiple-choice questions in order to diagnosis a specific illness or condition or recommend the user seek medical attention. The number of questions typically ranges from 10 to 30 before reaching a diagnosis, treatment advice, or a recommendation to seek medical attention. The answers to prior questions determine which questions are asked later as the system narrows down the possibilities. With the exception of three emergency questions determining whether an ambulance is needed urgently, the series of questions forms a tree. Using this property we were able to fully crawl every series of answers in the entire application.

We performed two sets of analysis for the NHS tool. The first is the subtree of the questionnaire for an adult male, the largest subtree after answering one's gender and age (468 states). We choose to do this in addition to the entire symptom checker for the subtree's interesting results and illustrate the power an attacker gains when starting with just two basic pieces of known context. The complete NHS symptom tracker has over 7300 paths through the questionnaire, each revealing different information about the user. Figure 9 summarizes the results.

To a much greater degree than the other web applications, the different threat models greatly affect classification performance. For example, using the Total-Source-Destination is significantly more effective in the ISP scenario than in the WiFi scenario. Loading full web pages, unlike the simple AJAX requests in the other applications, causes a significantly greater amount of noise due to TCP features such as ACKs and retransmissions. The inability to identify and filter out TCP features in the WiFi scenario greatly reduces classifiability for the Total-Source-Destination and Size-Weighted-Edit-Distance.

Edit-Distance Anomaly. Note that the Edit-Distance metric under the WiFi threat model performs better than under the ISP threat model. All metrics under in the ISP scenario ignore TCP features such as ACKs because they are easily manipulated either through padding the payload of ACK, by padding the transfers or by changing the TCP window size which indirectly manipulates the number of ACKs that will be sent. In the WiFi scenario, the attacker cannot filter out faked TCP ACKs, making classification more difficult. However, in our experiments TCP ACKs were legitimate and so when they are left in the trace they server as an indicator of the size of the transfer, and not random noise as would be expected in a strong defense system.

6.4 HTTPOS Defense

HTTPOS is a client-side defense against these attacks which substantially manipulates browser traffic to protect against analysis [22]. We deployed a prototype version of HTTPOS in the Firefox browser running our tests and measured its effectiveness. HTTPOS simply acts a SOCKS proxy and so we configured Firefox 3.6.17 to direct traffic through the HTTPOS system. We tested HTTPOS on Google search suggestions and Google Instant search, without a training phase and with all defenses enabled. The ability to easily apply and evaluate a previously published defense shows the flexibility of our system and the utility of the black-box approach for defense quantification and comparison.

The results of our initial tests are shown in Table 5. For search suggestions, HTTPOS is very effective. It significantly reduces the accuracy of our classifier across all metrics, resulting in performance only slightly better than random classification. However, in our experiments HTTPOS did not sufficiently mitigate the side-channel for Google Instant search. The accuracy using the TSD metric remained over 40%, which combined with successive letters

in a search query, we still believe the tool to remain exploitable. This is due to the much greater variation in different flows found in a Google Instant search due to integration of images, embedded maps, and videos. Without a proper training phase HTTPOS is unaware of the degree of traffic manipulation necessary to suppress the leak. Also noteworthy is the increase in Fisher criterion values for the SWED and ED metrics under the ISP threat model. We were not able to identify the specific HTTPOS defense mechanism that causes this increase, but we advocate any new defense mechanisms be throughly tested for accidentally created side-channels. Taken together, these experiments validate the ability of HTTPOS to effectively manipulate network traffic to thwart side-channel attacks on simple flows, but for complex flows and pages a training phase is required. Such a restriction reduces the utility and real-world applicability of the defense, but the effectiveness of HTTPOS for the search suggestions shows that a generic client-side defense is still promising for many applications.

6.5 MLComp

MLComp (http://mlcomp.org) is a service for comparing machine learning algorithms on shared datasets. Users upload programs and data sets in a standard format, allowing others to test their algorithms against a variety of data sets, or choose good classification techniques for their datasets. We used MLComp to compare our classifiers with the best available generic classifiers on the site. Table 6 summarizes the accuracy of the best classifier for each dataset. As expected, the results generally show worse performance than our classifiers which are designed using domain-specific background knowledge, but in every case the best generic classifiers perform no less than 15% worse than ours. Larger datasets, such as Yahoo search, did not finish in the site's maximum computation time of 24-hours so are not included in this table.

7. CONCLUSION

Side-channel leaks of private data have been found in popular web applications. Without tools to precisely quantify the leaks, developers cannot eliminate side-channel leaks without also sacrificing the responsiveness expected of modern web applications. Our detection system infers a web application state machine only using network traffic and the browser DOM. Our dynamic, black-box approach allows us to experiment and identify side-channel vulnerabilities in real-world web applications without access to source code. The Fisher criterion metric we propose is able to estimate the severity of application leaks much more accurately than is possible with entropy-based metrics. We have demonstrated the applicability of our approach by performing side-channel vulnerability analysis on large systems. Mitigating side-channel leaks remains an elusive goal, but our results provide encouraging evidence the side-channel leaks can be found automatically in a robust way.

Availability

Our crawling framework and quantification tool is available under an open source license from http://www.cs.virginia.edu/sca.

Acknowledgments

The authors thank Shuo Chen and XiaoFeng Wang for introducing us to the interesting problem of web application side-channel leaks. We thank Daniel Xiapu Luo for generously providing us with an early version of HTTPOS. This material is based upon work partly supported by grants from the National Science Foundation and by the Air Force Office of Scientific Research under MURI award FA9550-09-1-0539.

8. REFERENCES

[1] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic Side-Channel Attacks on Printers. In *19th USENIX Security Symposium*, 2010.

[2] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the Art: Automated Black-Box Web Application Vulnerability Testing. In *31st IEEE Symposium on Security and Privacy*, 2010.

[3] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies Workshop*, 2005.

[4] Mike Bowler. HtmlUnit. http://htmlunit.sourceforge.net/.

[5] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow. In *31st IEEE Symposium on Security and Privacy*, 2010.

[6] Heyning Cheng and Ron Avnur. Traffic Analysis of SSL Encrypted Web Browsing. UC Berkeley CS 261 Final Report, http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps, 1998.

[7] James Clark and Steve DeRose. XML Path Language (XPath). http://www.w3.org/TR/xpath/, 1999.

[8] George Danezis. Traffic Analysis of the HTTP Protocol over TLS. http://research.microsoft.com/en-us/um/people/gdane/papers/TLSanon.pdf, 2009.

[9] Al Danial. CLOC: Count Lines of Code. http://cloc.sourceforge.net/, 2006–2011.

[10] Roy T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[11] Ronald A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 1936.

[12] Flash-Selenium Project. A Selenium Extension for Enabling Selenium to Test Flash Components, 2011.

[13] Jeffrey Friedman. TEMPEST: A Signal Problem. *Cryptologic Spectrum*, 2007.

[14] Keita Fujii. Jpcap. http://netresearch.ics.uci.edu/kfujii/jpcap/doc/.

[15] Grig Gheorghiu. A Look at Selenium. *Better Software*, 2005.

[16] Google. Google Web Toolkit. http://code.google.com/webtoolkit/.

[17] Hong Guo, Qing Zhang, and Asoke K. Nandi. Feature Generation Using Genetic Programming Based on Fisher Criterion. In *15th European Signal Processing Conference*, 2007.

[18] William G. J. Halfond and Alessandro Orso. Improving Test Case Generation for Web Applications using Automated Interface Discovery. In *6th Joint European Software Engineering Conference and ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2007.

[19] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *16th Annual Conference on Advances in Cryptology*, 1996.

[20] Mark Levene and George Loizou. Computing the Entropy of User Navigation in the Web. *International Journal of Information Technology and Decision Making*, 1999.

[21] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *13th ACM Conference on Computer and Communications Security*, 2006.

[22] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdiscio. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Network and Distributed System Security Symposium*, 2011.

[23] Atif M. Memon, Ishan Banerjee, and Adithya Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In *10th Working Conference on Reverse Engineering*, 2003.

[24] Ali Mesbah. Crawljax. http://crawljax.com/, 2008–2011.

[25] Ali Mesbah, Engin Bozdag, and Arie van Deursen. Crawling AJAX by Inferring User Interface State Changes. In *Eighth International Conference on Web Engineering*, 2008.

[26] Chunyan Mu and David Clark. Quantitative Analysis of Secure Information Flow via Probabilistic Semantics. In *International Conference on Availability, Reliability and Security*, 2009.

[27] Linda Dailey Paulson. Building Rich Web Applications with Ajax. *Computer*, October 2005.

[28] Selenium Issues. Clearing the Cache from Firefox Driver. http://code.google.com/p/selenium/issues/detail?id=548, 2010.

[29] Selenium Project. Selenium. http://seleniumhq.org/, 2004–2011.

[30] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *23rd IEEE Symposium on Security and Privacy*, 2002.

[31] Yong Xu and Guangming Lu. Analysis On Fisher Discriminant Criterion And Linear Separability Of Feature Space. In *International Conference on Computational Intelligence and Security*, 2006.

[32] Bing-Yi Zhang, Ya-Min Sun, Yu-Lan Bian, and Hong-Ke Zhang. Linear Discriminant Analysis in Network Traffic Modeling: Research Articles. *International Journal on Communication Systems*, February 2006.

[33] Kehuan Zhang, Zhou Li, Rui Wang, XiaoFeng Wang, and Shuo Chen. Sidebuster: Automated Detection and Quantification of Side-Channel Leaks in Web Application Development. In *17th ACM Conference on Computer and Communications Security*, 2010.