Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow

Shuo Chen
Microsoft Research
Microsoft Corporation
Redmond, WA, USA
shuochen@microsoft.com

Rui Wang, XiaoFeng Wang, Kehuan Zhang
School of Informatics and Computing
Indiana University Bloomington
Bloomington, IN, USA
[wang63, xw7, kehzhang]@indiana.edu

***Abstract*– With software-as-a-service becoming mainstream, more and more applications are delivered to the client through the Web. Unlike a desktop application, a web application is split into browser-side and server-side components. A subset of the application's internal information flows are inevitably exposed on the network. We show that despite encryption, such a side-channel information leak is a realistic and serious threat to user privacy. Specifically, we found that surprisingly detailed sensitive information is being leaked out from a number of high-profile, top-of-the-line web applications in healthcare, taxation, investment and web search: an eavesdropper can infer the illnesses/medications/surgeries of the user, her family income and investment secrets, despite HTTPS protection; a stranger on the street can glean enterprise employees' web search queries, despite WPA/WPA2 Wi-Fi encryption. More importantly, the root causes of the problem are some fundamental characteristics of web applications: stateful communication, low entropy input for better interaction, and significant traffic distinctions. As a result, the scope of the problem seems industry-wide. We further present a concrete analysis to demonstrate the challenges of mitigating such a threat, which points to the necessity of a disciplined engineering practice for side-channel mitigations in future web application developments.**

Keywords*– *side-channel-leak; Software-as-a-Service (SaaS); web application; encrypted traffic; ambiguity set; padding

I. INTRODUCTION

Regarding the pseudonyms used in the paper
This paper reports information leaks in several real-world web applications. We have notified all the affected parties of our findings. Some requested us to anonymize their product names. Throughout the paper, we use superscript "A" to denote such pseudonyms, e.g., *OnlineHealth*^{A}, *OnlineTax*^{A}, and *OnlineInvest*^{A}.

The drastic evolution in web-based computing has come to the stage where applications are increasingly delivered as services to web clients. Such a software-as-a-service (SaaS) paradigm excites the software industry. Compared to desktop software, web applications have the advantage of not requiring client-side installations or updates, and thus are easier to deploy and maintain. Today web applications are widely used to process very sensitive user data including emails, health records, investments, etc. However, unlike its desktop counterpart, a web application is split into browser-side and server-side components. A subset of the application's internal information flows (i.e., data flows and control flows) are inevitably exposed on the network, which may reveal application states and state-transitions. To protect the information in critical applications against network sniffing, a common practice is to encrypt their network traffic. However, as discovered in our research, serious information leaks are still a reality.

For example, consider a user who enters her health profile into *OnlineHealth*^{A} by choosing an illness condition from a list provided by the application. Selection of a certain illness causes the browser to communicate with the server-side component of the application, which in turn updates its state, and displays the illness on the browser-side user interface. Even though the communications generated during these state transitions are protected by HTTPS, their observable attributes, such as packet sizes and timings, can still give away the information about the user's selection.

Side-channel information leaks. It is well known that the aforementioned attributes of encrypted traffic, often referred to as *side-channel information*, can be used to obtain some insights about the communications. Such side-channel information leaks have been extensively studied for a decade, in the context of secure shell (SSH) [15], video-streaming [13], voice-over-IP (VoIP) [23], web browsing and others. Particularly, a line of research conducted by various research groups has studied anonymity issues in encrypted web traffic. It has been shown that because each web page has a distinct size, and usually loads some resource objects (e.g., images) of different sizes, the attacker can fingerprint the page so that even when a user visits it through HTTPS, the page can be re-identified [7][16]. This is a concern for anonymity channels such as Tor [17], which are expected to hide users' page-visits from eavesdroppers.

Although such side-channel leaks of web traffic have been known for years, the whole issue seems to be neglected by the general web industry, presumably because little evidence exists to demonstrate the seriousness of their consequences other than the effect on the users of anonymity channels. Today, the Web has evolved beyond a publishing system for static web pages, and instead, becomes a platform for delivering full-fledged software applications. The side-channel vulnerabilities of encrypted communications, coupled with the distinct features of web applications (e.g., stateful communications) are becoming an unprecedented threat to the confidentiality of user data processed by these applications, which are often far more sensitive than the identifiability of web pages studied in the prior anonymity research. In the OnlineHealth^A example,

different health records correspond to different state-transitions in the application, whose traffic features allow the attacker to effectively infer a user's health information. Despite the importance of this side-channel threat, little has been done in the web application domain to understand its scope and gravity, and the technical challenges in developing its mitigations.

Our work. In this paper, we report our findings on the magnitude of such side-channel information leaks. Our research shows that surprisingly detailed sensitive user data can be reliably inferred from the web traffic of a number of high-profile, top-of-the-line web applications such as OnlineHealth[A], OnlineTax[A] Online, OnlineInvest[A] and Google/Yahoo/Bing search engines: an eavesdropper can infer the medications/surgeries/illnesses of the user, her annual family income and investment choices and money allocations, even though the web traffic is protected by HTTPS. We also show that even in a corporate building that deploys the up-to-date WPA/WPA2 Wi-Fi encryptions, a stranger without any credential can sit outside the building to glean the query words entered into employees' laptops, as if they were exposed in plain text in the air. This enables the attacker to profile people's actual online activities.

More importantly, we found that the root causes of the problem are certain pervasive design features of Web 2.0 applications: for example, AJAX GUI widgets that generate web traffic in response to even a single keystroke input or mouse click, diverse resource objects (scripts, images, Flash, etc.) that make the traffic features associated with each state transition distinct, and an application's stateful interactions with its user that enable the attacker to link multiple observations together to infer sensitive user data. These features make the side-channel vulnerability fundamental to Web 2.0 applications.

Regarding the defense, our analyses of real-world vulnerability scenarios suggest that mitigation of the threat requires today's application development practice to be significantly improved. Although it is easy to conceive high-level mitigation strategies such as packet padding, concrete mitigation policies have to be specific to individual applications. This need of case-by-case remedies indicates the challenges the problem presents: on one hand, detection of the side-channel vulnerabilities can be hard, which requires developers to analyze application semantics, feature designs, traffic characteristics and publicly available domain knowledge. On the other hand, we show that without finding the vulnerabilities, mitigation policies are likely to be ineffective or incur prohibitively high communication overheads. These technical challenges come from the fact that sensitive information can be leaked out at many application states due to the stateful nature of web applications, and at different layers of the SaaS infrastructure due to its complexities. Therefore, effective defense against the side-channel leaks is a future research topic with strong practical relevance.

In addition, we realized that enforcing the security policies to control side-channel leaks should be a joint work by web application, browser and web server. Today's browsers and web servers are not ready to enforce even the most basic policies, due to the lack of cross-layer communications, so we designed a side-channel control infrastructure and prototyped its components as a *Firefox* add-on and an *IIS* extension, as elaborated in Appendix C.

Contributions. The contributions of this paper are summarized as follows:

- *Analysis of the side-channel weakness in web applications.* We present a model to analyze the side-channel weakness in web applications and attribute the problem to prominent design features of these applications. We then show concrete vulnerabilities in several high-profile and really popular web applications, which disclose different types of sensitive information through various application features. These studies lead to the conclusion that *the side-channel information leaks are likely to be fundamental to web applications*.
- *In-depth study on the challenges in mitigating the threat.* We evaluated the effectiveness and the overhead of common mitigation techniques. Our research shows that effective solutions to the side-channel problem have to be application-specific, relying on an in-depth understanding of the application being protected. This suggests the necessity of *a significant improvement of the current practice for developing web applications*.

Roadmap. The rest of the paper is organized as follows: Section II surveys related prior work and compares it with our research; Section III describes an abstract analysis of the side-channel weaknesses in web applications; Section IV reports such weaknesses in high-profile applications and our techniques that exploit them; Section V analyzes the challenges in mitigating such a threat and presents our vision on a disciplined development practice for future web applications; Section VI concludes the paper.

II. Related Work

Side channel leaks have been known for decades. A documented attack is dated back to 1943 [22]. Side-channel leaks are discussed broadly in many contexts, not necessarily about encrypted communications. Information can be leaked through electromagnetic signals, shared memory/registers/files between processes, CPU usage metrics, etc. Researchers have shown that keystroke recoveries are feasible due to keyboard electromagnetic emanations [18]. In Linux, the stack pointer ESP of a process can be profiled by an attack process, and thus inter-keystroke timing information can be estimated in the cross-process manner [24]. Also related is the research on the co-resident-VM problem within commercial cloud computing infrastructures: Ristenpart et al demonstrated that an Amazon EC2 user can intentionally place a VM on the same

physical machine as another customer's VM, which allows the former to estimate the cache usage, traffic load and keystroke timing of the latter [12].

In the context of encrypted communications, it has been shown that the side-channel information, such as packet timing and sizes, allows a network eavesdropper to break cryptographic systems or infer keystrokes in SSH, spoken phrases in VoIP and movie titles in video-streaming systems. Brumley et al showed a timing attack against OpenSSL that extracts RSA secret keys [2]. Song et al showed that because SSH is an interactive remote shell service and typing different keystroke-combinations naturally produces slight timing characteristics, a network eavesdropper can build a Hidden Markov Model (HMM) to infer the keystrokes [15]. When applied to guess a password, the attack achieves a 50-time speedup compared to a brute-force guessing attack, i.e., more than 6-bit reduction of the password's entropy. Wright et al studied the side-channel leak in Voice-over-IP systems that use variable-bit-rate encoding schemes [23]. In their experiment, simulated conversations were constructed by randomly selecting sentences from a standard corpus containing thousands of spoken sentences. They tried to determine if a target sentences, also from the corpus, exists in each conversation, and achieved 0.5 recall and 0.5 precision, i.e., when a target sentence is in a conversation, the attack algorithm says yes with a 0.5 probability; when the attack algorithm says yes, there is a 0.5 probability that the target sentence is in the conversation. Saponas et al showed that the side-channel leak from *Slingbox Pro*, a device for encrypted video-streaming, allows the attacker to determine the title of the movie being played [13].

In the context of encrypted web communications, researchers have recognized the web anonymity issue for many years, i.e., the attacker can fingerprint web pages by their side-channel characteristics, then eavesdrop on the victim user's encrypted traffic to identify which web pages the user visits. Wagner and Schneier briefly cited their personal communication with Yee in 1996 about the possibility of using this idea against SSL/TLS [19]. An actual attack demo was described in a course project report in 1998 by Cheng et al [6]. Sun et al [16] and Danezis [7] both indicated that this type of side-channel attack defeats the goal of anonymity channels, such as Tor, MixMaster and WebMixes. Sun et al's experiment showed that 100,000 web pages from a wide range of different sites could be effectively fingerprinted. Besides SSL/TLS, Bissias et al conducted a similar experiment on WPA and IPSec [4].

Our work is motivated by these anonymity studies, but is different in a number of major aspects: (1) our study focuses on web applications and the sensitive user data leaked out from them, rather than the identifiability of individual web pages; (2) application state-transitions and semantics are the focal point of our analyses, while the prior studies are agnostic to them; (3) our target audience is the developers of sensitive web applications, while the natural audience of the web-anonymity research is the providers of anonymity channels, as their objective is directly confronted by the anonymity issue studied in the prior research.

III. Fundamentals of Web Application Information Leaks

Conceptually, a web application is quite similar to a traditional desktop application. They both work on input data from the user or the file system/database, and their state-transitions are driven by their internal information flows (both data flows and control flows). The only fundamental difference between them is that a web application's input points, program logic and program states are split between the browser and the server, so a subset of its information flows must go through the network. We refer to them as *web flows*. Web flows are subject to eavesdropping on the wire and in the air, and thus often protected by HTTPS and Wi-Fi encryptions.

The attacker's goal is to infer sensitive information from the encrypted web traffic. In other words, an attack can be thought of as an *ambiguity-set reduction* process, where the ambiguity-set of a piece of data is the set containing all possible values of the data that are indistinguishable to the attacker. How effectively the attacker can reduce the size of the ambiguity-set quantifies the amount of information leaked out from the communications – if the ambiguity-set can be reduced to $1/\Re$ of its original size, we say that $log_2\Re$ *bits of entropy* of the data are lost. Similar modeling of inference attack was also discussed in prior research, for example, elimination of impossible traces in [8].

Following we present a model of web applications and their side-channel leaks. The objective is to make explicit the key conditions under which application data can be inferred. We then correlate these conditions to some pervasive properties of web applications.

A. Model Abstraction

A web application can be modeled as a quintuple $(S, \Sigma, \delta, f, V)$, where S is a set of program states that describe the application data both on the browser, such as the DOM (Document Object Model) tree and the cookies, and on the web server. Here we treat back-end databases as an external resource to a web application, from which the application receives inputs. Σ is a set of inputs the application accepts, which can come from the user (e.g., keystroke inputs) or back-end databases (e.g., the account balance). A transition from one state to another is driven by the input the former receives, which is modeled as a function δ: $S \times \Sigma \rightarrow S$. A state transition in our model always happens with web flows, whose observable attributes, such as packet sizes, number of packets, etc., can be used to characterize the original state and its inputs. This observation is modeled as a function f: $S \times \Sigma \rightarrow V$, where V is a set of web flow vectors that describe the observable characteristics of the encrypted traffic. A *web flow vector* v is a sequence of directional packet sizes, e.g., a

50-byte packet from the browser and a 1024-byte packet from the server are denoted by "(50➔, ➔1024)".

B. *Inference of Sensitive Inputs*

The objective of the adversary can be formalized as follows. Consider at time t an application state s_t to accept an input (from the user or the back-end database). The input space is partitioned into k semantically-disjoined sets, each of which brings the application into a distinct state reachable from s_t. For example, family incomes are often grouped into different income ranges, which drive a tax preparation application into different states for different tax forms. All k such subsequent states form a set $S_{t+1} \subset S$. The attacker intends to figure out the input set containing the data that the application receives in s_t, by looking at a sequence of vectors (v_t , v_{t+1}, … , v_{t+n-1}) caused by n consecutive state transitions initiated from s_t . This process is illustrated in Figure 1. It is evident that a solution to this problem can be applied recursively, starting from s_0, to infer the sensitive inputs of the states that the web application goes through.

Before observing the vector sequence, the attacker has no knowledge about the input in s_t: all the k possible input sets constitute an ambiguity set of size k. Upon seeing v_t, the attacker knows that only transitions to a subset of S_{t+1}, denoted by D_{t+1}, can produce this vector, and therefore infers that the actual input can only come from k/α sets in the input space, where $\alpha \in [1,k)$ is the *reduction factor* of this state transition. The new ambiguity set D_{t+1} can further be reduced by the follow-up observations (v_{t+1}, … , v_{t+n-1}). Denote the ratio of this reduction by β, where $\beta \in [1,\infty)$. In the end, the attacker is able to identify one of the $k/(\alpha\beta)$ input sets, which the actual input belongs to.

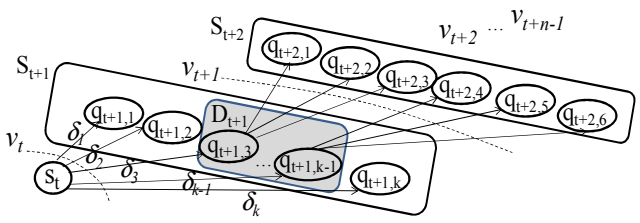

Figure 1: Ambiguity set reduction

C. *Threat Analysis over Web Application Properties*

The above analysis demonstrates the feasibility of side-channel information leaks in web applications. The magnitude of such a threat to a specific web application, however, depends on the size of the input space of the sensitive data and the reduction factors incurred by its state transitions. The former determines whether it is possible for the attacker to efficiently test input values to identify those that produce the web traffic matching the observed attribute vectors. The latter indicates the amount of the information the attacker can learn from such observations. In this section, we show that some prominent features of today's web application design often lead to low entropy inputs and large reduction factors, making the threat realistic.

Low entropy input for better interactions. State transitions of a web application are often caused by the input data from a relatively small input space. Such a low-entropy input often come as a result of the increasing use of highly interactive and dynamic web interfaces, based upon the techniques such as AJAX (asynchronous JavaScript and XML). Incorporation of such techniques into the GUI widgets of the application makes it highly responsive to user inputs: even a single mouse click on a check box or a single letter entered into a text box could trigger web traffic for updating some DOM objects within the application's browser-side interface. Examples of such widgets include *auto-suggestion* or *auto-complete* that populates a list of suggested contents in response to every letter the user types into a text box, and asynchronously updating part of the HTML page according to every mouse click. Such widgets have been extensively used in many popular web applications hosted by major web content providers like Facebook, Google and Yahoo. They are also supported by mainstream JavaScript libraries for web application development: Appendix A lists 14 such libraries. Moreover, the interfaces of web applications are often designed to guide the user to enter her data step by step, through interacting with their server-side components. Those features cause the state transitions within a web application to be triggered by even a very small amount of input data, and as a result, enable the attacker to enumerate all possible input values to match the observed web flow vector.

The user data that a web application reads from its back-end database can also be low entropy: for example, the image representations of some types of user data have only enumerable possibilities. This can result in disclosure of sensitive user information, such as the mutual fund choices of one's investment, as elaborated in Section IV.C.

Stateful communications. Like desktop applications, web applications are stateful: transitions to next states depend both on the current state and on its input. To distinguish the input data in Figure 1, the attacker can utilize not only v_t but also every vector observed along the follow-up transition sequences. This increases the possibility of distinguishing the input. For example, a letter entered in a text box affect all the follow-up auto-suggestion contents, so the attributes of the web traffic (for transferring such contents) associated with both the current letter and its follow-up inputs can be used to infer the letter. Although the reduction factor for each transition may seem insignificant, the combination of these factors, which is application-specific, can be really powerful. We will show through real application scenarios that such reduction powers are often multiplicative, i.e., $\beta = \beta_{t+1} \bullet \ldots \bullet \beta_{t+n}$, where β_x is the reduction factor achieved by observing vector v_x.

Significant traffic distinctions. Ultimately the attacker relies on traffic distinctions to acquire the reduction factor from each web flow. Such distinctions often come from the objects updated by browser-server data exchanges, which

usually have highly disparate sizes. As an example, we collected image objects, HTML documents and JavaScript objects from five popular websites, and studied the distributions of their sizes. The outcome, as presented in Table I, shows that the sizes of the objects hosted by the same website are so diverse that their standard deviations (σ) often come close or even exceed their means (μ).

Table I. SIZES OF OBJECTS ON FIVE POPULAR WEBSITES

	JPEG		HTML code		Javascript	
(In bytes)	μ	σ	μ	σ	μ	σ
cnn.com	5385	7856	73192	25862	6453	6684
health.state.pa.us	12235	7374	49917	10591	N/A	N/A
medicineNet.com	3931	2239	49313	14472	22530	28184
nlm.nih.gov	11918	48897	22581	15430	4934	5307
WashingtonPost .com	12037	15122	90353	35476	13413	36220

On the other hand, cryptographic protocols like HTTPS, WPA and WPA2, cannot cover such a large diversity. We will explain later that WPA/WPA2 do not hide packet sizes at all. HTTPS allows websites to specify ciphers. If a block cipher is used, packet sizes will be rounded-up to a multiple of the block size. We checked 22 important HTTPS websites in Appendix B. All of them use RC4 stream cipher, except two: VeriSign, which uses AES128 block cipher for some communications and RC4 for others, and GEICO, which uses Triple-DES block cipher (64 bits). No AES256 traffic was observed on any website. This indicates that the vast majority of the websites adopts RC4, presumably because it is considerably faster than block ciphers. Note that we simply state the fact that most websites today have absolutely no side-channel protection at the HTTPS layer, not advocating block ciphers as a cure. We will show later that for most application features, the rounding-effects of block ciphers offer very marginal or no mitigation at all because the traffic distinctions are often too large to hide.

In Section IV, we use a metric *density* to describe the extent to which packets can be differentiated by their sizes. Let $\wp$ be a set of packet sizes. We define density($\wp$) = $|\wp|$ / [max($\wp$)-min($\wp$)], which is the average number of packet(s) for every possible packet size. A density below 1.0 often indicates that the set of packets are easy to distinguish.

Summary. The above analysis shows that the root cause of the side-channel vulnerability in web applications are actually some of their fundamental features, such as frequent small communications, diversity in the contents exchanged in state transitions, and stateful communications. Next, we describe the problem in real-world applications.

IV. ACTUAL INFORMATION LEAKS IN HIGH-PROFILE APPLICATIONS

As discussed in the previous section, some pervasive design features render web applications vulnerable to side-channel information leaks. This section further reports our study on the gravity of the problem in reality, through analyzing the side-channel weaknesses in a set of high-profile web applications.

We found that these applications leak out private user data such as health information, family income data, investment secrets and search queries. Both user surveys and real life scenarios show that people treat such data as highly confidential. For example, a study conducted by BusinessWeek "*confirms that Americans care deeply about their privacy. … 35% of people would not be at all comfortable with their online actions being profiled, but 82% are not at all comfortable with online activities being merged with personally identifiable information, such as your income, driver's license, credit data, and medical status* [5]." In another survey, which was about sex practices in the U.S. (the topic in itself was sensitive), the respondents identified family income as the most sensitive question in the survey [1]. Besides the public perception reported by those surveys, the impact of such information can also be observed in real life. For example, the public was concerned about the true health condition of a big company's CEO. It is thought that his health matter could affect the company's stock price by 20%-25% [21]. Similarly, details of fund holdings are secret information of big investors: for example, a major hedge fund management firm was reported to worry that the government's auditing might leak out its investment strategies and hurt its competitive edge [11].

In the rest of this section, we elaborate how such information is leaked out from these leading applications. Before we come to the details of our findings, it is important to notice that identification of a running web application remotely can be practically achieved through de-anonymizing web traffic [16] [7]. When Ethernet sniffing is possible, the application can usually be easily identified by `nslookup` using its server's IP address.

A. *OnlineHealth*^{A}

OnlineHealth^A is a personal health information service. It is developed by one of the most reputable companies of online services. OnlineHealth^A runs exclusively on HTTPS. Once logged in, a user can build her health profile by entering her medical information within several categories, including Conditions, Medications, Procedures, etc. The user can also find doctors with different specialties. In our research, we constructed an attack program to demonstrate that an eavesdropper is able to infer the medications the user takes, the procedures she has, and the type of doctors she is looking for.

1) "Add Health Records"

One of the main functionalities of OnlineHealth^A is to add various types of health records. Figure 2 illustrates the user interface. On the top of the page are the tabs that specify the types of the records to be entered. In the figure, the tab "Conditions" has been selected, which allows the user to input a condition (i.e., symptom/illness). The record can be entered through typing, which is assisted by an auto-

suggestion widget, or by mouse selection. Other types of records can be entered in the similar way.

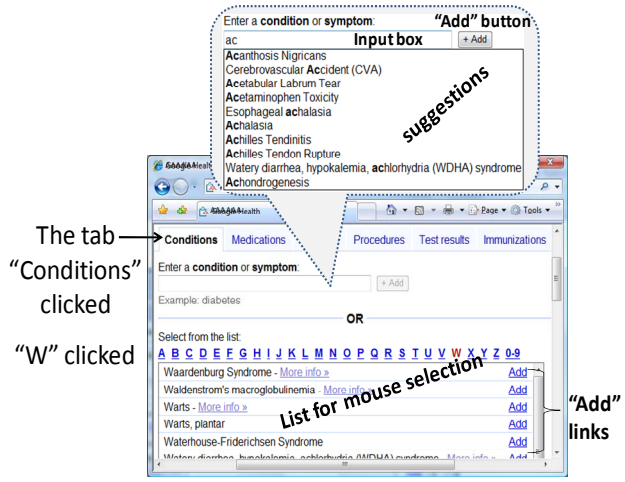

Figure 2: User interface for adding health records

This tab design already leaks out information about the type of the record being added because every tab click generates a web flow vector (1515±1➔, 266±1➔, ➔583±1, ➔x), where x takes 4855, 30154, 20567, 1773, 2757 and 2299, for Conditions, Medications, Allergies, Procedures, Test Results and Immunizations, respectively. The density of the tab-clicks is 6/(30154-1773)= 0.000211. (Note that some packets in the vector have small deviations for different user accounts, so we use symbol ± to denote them.) Once a tab has been selected, the user further interacts with the application in one of the following two ways.

Input by typing – the caveat of auto suggestion. As the user types, a suggestion list appears under the input box. The list contains at most ten items and is updated in response to every keystroke[1]. Figure 2 shows the list after the user types "ac" in the box. The user can continue typing, or select one of the items from the list and click "Add".

Interestingly, the auto-suggestion in fact causes a catastrophic leak of user input, because the attacker can effectively disambiguate the user's actual input after every keystroke by matching the size of the response carrying the suggestion list. More specifically, every keystroke generates a web flow vector (253±1➔, ➔581, ➔x), where x precisely indicates the size of the suggestion list, and is same across all users (i.e., the attacker and the victims).

The communications are stateful: each keystroke produces a web flow determined by not only the current letter being typed but also all other letters entered prior to it (i.e., its prefix). As discussed in Section III.B, such stateful communications enable the inference of the input. Of course, the effectiveness of such an inference depends on the reduction factors α and β. To get a sense about α, we collected the 26 x-values when typing "a" to "z" as the first character in the input box under the Conditions tab. The values are in the range of [273,519], i.e., density=0.11. All the values are distinct, except the letters "h" and "m", which produce the same x value. Similarly, we collected the x-values when entering "a" .. "z" posterior to an "a". Only 20 such combinations brings in non-empty suggestion lists (the others are invalid), and their x-values appear within the range of [204,515], i.e., density=0.064. These tests show that the reduction factor α is significant.

The factor β, representing the information leaks caused by the follow-up state transitions, further helps reduce the ambiguity set of the input. For example, the letters "h" and "m", when entered as the first letter in the text box, cannot be differentiated by the eavesdropper immediately, as they all produce identical web flows. However, we collected the x-values for "ha" to "hz" and "ma" to "mz". Among these 52 strings, only 20 are valid. All x-values are distinct except "ha" and "ma". The x-value range is [213,434], i.e., density=0.090. Therefore, by observing the web flow of the second keystroke, the first letter "h" and "m" can be effectively disambiguated (except "ha" and "ma").

After entering some letters, the user can select a suggestion and click the "Add" button to submit the request. The web flow we observed is (x➔, ➔580, ➔53±1), which also contributes to the reduction power. For example, the x-values of the ten suggested items for the input "*head*" fall in [1185, 1283] with the density 0.10, and no collision.

Under the tabs other than "Conditions", we made the similar observations, which indicate that the user has almost no secret when entering her records through typing.

Input by mouse selecting – a caveat of hierarchical organization of user choices. Alternatively, the user can use mouse clicks to add a record: through choosing a tab, then a character in the alphabetical list (26 letters and the link marked as "0-9"), and finally the "Add" link of an item (see Figure 2). This selection is essentially a stateful navigation on a tree-hierarchy. This design significantly lowers the entropy of the user input, and in the meantime makes the application states clearly identifiable.

Let us again use Conditions as an example. OnlineHealth^A has 2670 conditions, which are grouped by their initial characters: for example, the list in Figure 2 consists of the Conditions starting with W because W has been clicked. We collected the response size x when clicking every character. The range of x is [226, 5876], i.e., density=0.0046. It is trivial to identify the initial character the user clicks.

Figure 3 describes how the 2670 conditions are distributed among the characters. For each character, the bar shows the number of conditions whose names start with the character. We also show the density of the conditions under each character, e.g., A=2.50, J=0.08, etc. Many letters have their densities around or below 1. Even the highest one is only 2.5. Given the total number of conditions being 2670,

[1] This functionality has no cache effect, so the traffic is always observed on the network. We will discuss in Section I.A.2) that the cache effect would not fundamentally salvage the auto-suggestion leak even if it existed.

even the 2.5 density offers the attacker 1068× (i.e.,1068-time) reduction power.

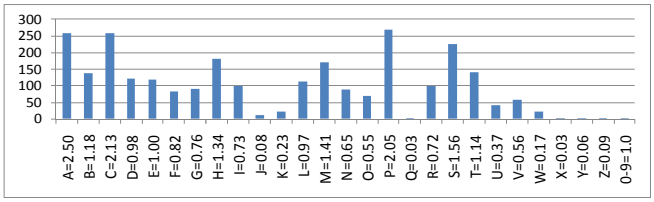

Figure 3: Number of Conditions under each character

On the tabs other than Conditions, we observed the similar traffic patterns, indicating that they are similarly subject to the side-channel attack.

2) "Find a Doctor"

Another useful feature of OnlineHealth^A is "find a doctor", as shown in Figure 4. By choosing a specialty from the drop-down list and entering a city name (or a zipcode), the user searches the database of OnlineHealth^A to get a list of doctors matching her desired specialty.

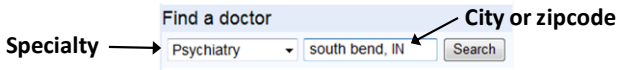

Figure 4: "Find a doctor" feature

We assume that a patient tends to find doctors near her current geographical location. Therefore the input of "city or zipcode" is guessable based on her IP address. When the "search" button is clicked, the web flow vector is (1507➔, 270±10➔, ←582±1, ←x). Selection from the drop-down list gives a very-low-entropy input: there are only 94 specialties. We tested all the specialties in "south bend, IN", and found that x was within [596, 1660], i.e., density = 0.089, and every specialty is uniquely identifiable.

B. *OnlineTax^A*

We studied OnlineTax^A, the online version of one of the most widely used applications for preparing the United States' tax documents (a.k.a. tax returns) for individuals and businesses. They are accessible through HTTPS exclusively. We found that the web applications leak a large amount of user information, such as family income, whether the user paid big medical bills, etc. The family income is particularly sensitive as discussed earlier.

1) Background

The U.S. taxpayers pay annual taxes – by April 15th, they file the tax returns for the previous year in order to claim back the money over-withheld by the government or pay the government any owed tax. The tax returns are a set of standard forms designed by the government. The total number of different forms is very big. Fortunately, depending on one's specific tax situation, she only needs to work on a subset of the forms. Although the tax laws explain which forms to file, it requires considerable brain power to understand the laws and accomplish the necessary calculations to make tax-return claims right.

OnlineTax^A is a tax-preparation application designed as a wizard that essentially implements the tax laws as an "algorithm". It asks the user simple questions, and tailors the future questions based on the user's earlier answers. When the user finishes the conversation, OnlineTax^A has all the information for preparing required tax forms. Also, it is intelligent enough to find hundreds of tax deductions and tax credits to make the user's tax as low as possible.

2) Workflow of the Tax Calculation

The tax calculation has a clear workflow: it starts with basic personal information, followed by the calculation of federal taxes, then the state taxes. Figure 5 shows the main modules of OnlineTax^A. Sub-modules under the Federal Taxes module are also shown in the figure.

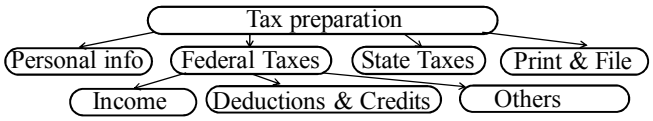

Figure 5: Main modules of OnlineTax^A and sub-modules of Federal Taxes

In the Personal Info module, the user enters the basic information about her family, e.g., the information about spouse and children. She also needs to select one of five filing statuses (to be explained later). These statuses profoundly affect the logic of the tax calculation.

After the Personal Info module is completed, the user starts to work on Federal Taxes. As shown in Figure 5, there are three sections: *Income* for reporting all types of incomes; *Deductions & Credits* for deducing the tax due according to available tax laws, and other tax situations.

The income information includes many categories, such as salaries, investments, etc. Based on the income information and the number of family members (a.k.a. exemptions), the Adjusted Gross Income (AGI) is calculated. AGI represents the family's actual income standing, and thus is the basis for many tax calculations.

3) Leaking Private Information

The attacker can easily learn certain basic facts from analyzing the application's traffic. For example, Figure 6 depicts the application's decision logic for the filing status. The state-transitions can be easily identified from the web flows of the application, as the web pages are very different.

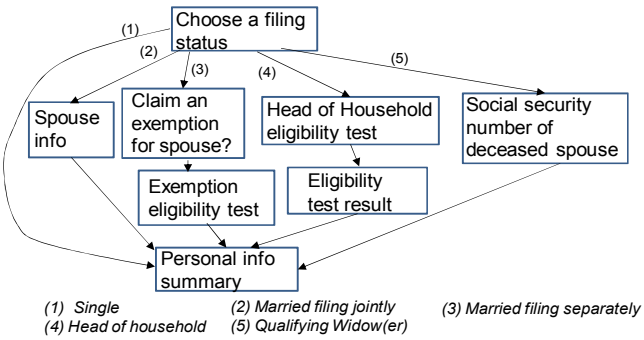

Figure 6: State machine that decides the filing status

Similarly, we can infer the number of children in the family by observing how many times the user fills out the child info page, and whether the spouse has salaries by observing the salary form submissions (a.k.a., W-2 form). All these user actions produce unique web flow vectors.

Inference of AGI. AGI is calculated by OnlineTax^{A} based on the data in the Income module. It is unlikely to infer the AGI from the sizes of W-2 or other income forms: for example, each W-2 form has twenty input boxes, most of which take the data of variable lengths; this gives the input (which includes the AGI) a sufficiently high entropy that discourages a side-channel analysis.

As described in Section III, communications in web applications are stateful. The inference of user input does not have to be accomplished when the data is initially entered by the user, and can instead make full use of the web flow vectors observed from the later state transitions that are also affected by the input data. When it comes to tax calculation, the dependencies between the AGI and tax deductions/credits become an intrinsic link to tie the AGI to the state transitions. More specifically, the attacker can utilize the following two facts: (1) the user's eligibilities for many credits and deductions depend on the AGI, and such dependencies can be identified from tax laws; (2) such eligibilities affect state transitions, which can be inferred from observed traffic patterns.

For example, Figure 7 shows OnlineTax^{A}'s state machine for determining one's eligibility for the *child credit*.

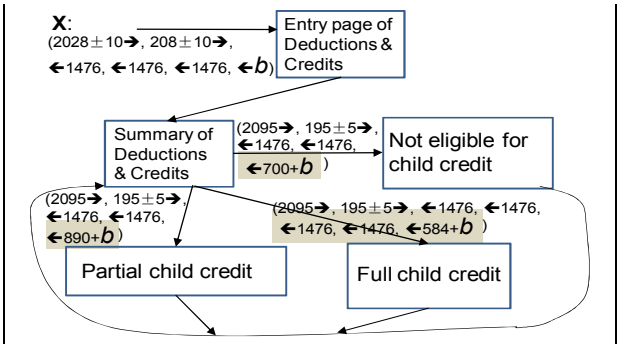

Figure 7: State transitions for child credit eligibilities

When the user gets the entry page of Deductions & Credits, web flow vector *X*: *(2028±10➔, 208±10➔, ←1476, ←1476, ←1476, ←b)* is observed, i.e., the attacker learns from the observation about the value of *b*, which is a value different for each user because the response packet contains user-profile data of a certain length. Based on the value of *b*, the attacker can determine the transition to one of the three possible states: "not eligible for child credit", "partial child credit" and "full child credit" (the web flow differences are highlighted in the figure). According to the tax law document IRS-Pub-792, (1) the taxpayer can claim up to \$1000*c credit, where c is the number of dependent children; (2) if the AGI is below \$110,000 for Married Filing Jointly, the taxpayer gets the full credit; (3) for every \$1000 income in the AGI above \$110,000, the taxpayer loses \$50 child credit (a.k.a. the phase-out rule). Therefore, if a taxpayer claims two children in the personal info module, by observing the transitions in Figure 7, the attacker is able to confidently identify where the taxpayer's AGI falls: below \$110,000, between \$110,000 and \$150,000, or above \$150,000.

Even more intriguing are other Credits & Deductions situations, which often have *asymmetric execution paths* for different eligibilities. For example, Figure 8 shows the Student Loan Interest deduction: if the AGI is higher than \$145,000, the user is not eligible, and no further question will be asked; otherwise, the application gets into the highlighted state where the information about the user's interest is required. Therefore, if the user is partially or fully qualified, the path of state transitions will be longer than that for those not eligible. This allows an eavesdropper to tell whether one's AGI is above \$145,000, even when the web flow vector of transition A in Figure 8 are made indistinguishable from that of D. Asymmetric paths widely exist in OnlineTax^{A}: all the nine credits and deductions listed in Figure 9, except "child credit" and "elderly or disabled credit", have this situation.

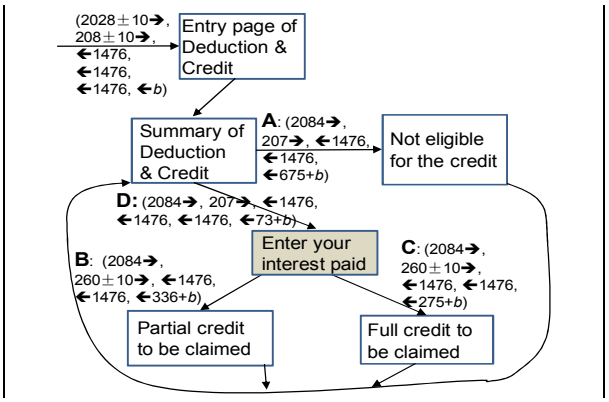

FIGURE 8: ASYMMETRIC PATHS IN STUDENT LOAN INTEREST DEDUCTION

We skimmed through the U.S. tax instructions to sample some common credits/deductions, and confirmed that the web flow vectors of OnlineTax^{A} indeed enable the attacker to infer a series of AGI ranges (see Figure 9) for those who attempt to claim the credits and deductions.

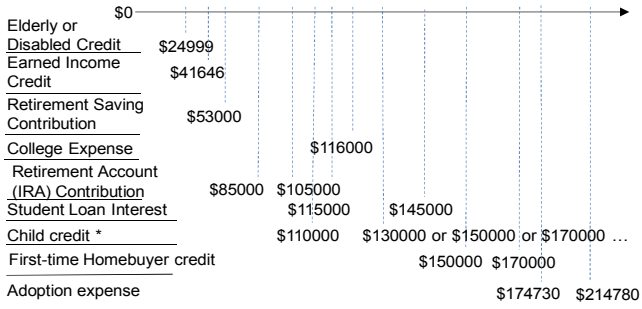

Figure 9: Some disclosed AGI ranges

So far we have only focused on the federal tax. Most U.S. states have their state income tax provisions. Many of them are also associated with the AGI. We believe that the same attack could work on state taxes. Since none of the authors of the paper is a tax expert, we only studied a small fraction of all (hundreds of) the credits and deductions that OnlineTax[A] can process. We believe that state transitions associated with those credits and deductions could give away more private information, such as other AGI ranges, whether the user paid large medical bills or mortgages, etc.

C. *OnlineInvest[A].com*

OnlineInvest[A] is a leading financial company in the U.S., which provides a wide range of financial products and services to investors. The service is also exclusively accessible through HTTPS. Different from prior examples in which data leaks are caused by auto-suggestion, search, and wizard conversation, etc, here privacy information is disclosed by graphical visualization of data.

1) The Mutual Fund Page and the Fund Allocation Page

When the user logs onto the OnlineInvest[A] account, one can choose to view investment holdings such as mutual funds, stocks or bond, in a list. Figure 10 (upper) illustrates an example in which the user invests three funds, and their 12-month price history charts are displayed on the side of the textual content. These charts are GIF images. Per OnlineInvest[A]'s request, we replaced the actual screenshots with the mock-up graphics.

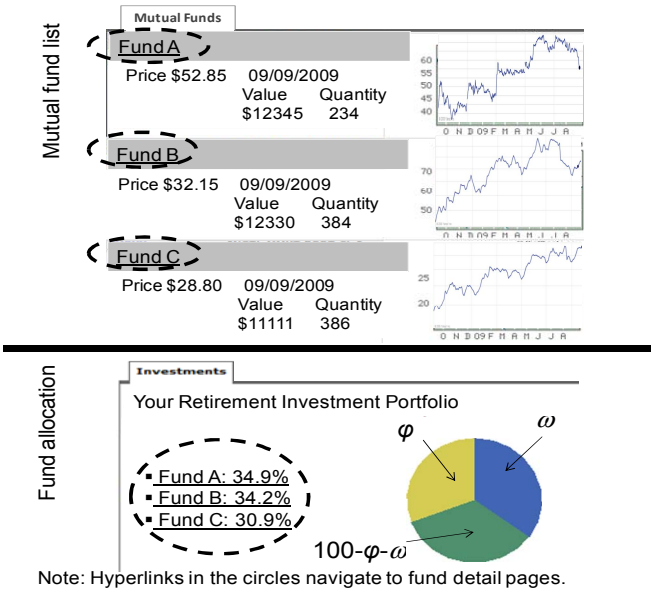

Figure 10: Mutual fund list (upper) and allocation (lower)

An image (i.e., <IMG>) on an HTML page is loaded separately from the page. Therefore the size of the image can be identified from the packet size of the response from the server. There are 9 mutual funds available in this type of account to choose, which are obviously a set of low entropy inputs. We recorded the sizes of all the price-history charts on 9/9/2009, which were all distinct, with a density 0.044. To access the fund details, the user can click the hyperlinks highlighted in circles in Figure 10. The sizes of the fund detail pages were all different too, with a much lower density 0.010.

Figure 10 (lower) is another view that can be selected by the user, which displays the percentages of the investment on the funds. The pie chart is also a GIF image. In the next subsection, we will see how the pie chart can be inferred from its size, which once again demonstrates the multiplicative reduction power that the attacker can obtain.

2) Leaking Investment Secrets

The price history charts and the fund detail pages are publicly accessible to everyone, with or without OnlineInvest[A] accounts (in fact, the browser imports the charts from the website *https://FinancialData[A].com*, which is not OnlineInvest[A]'s property). Since the choices of mutual funds have low entropy and the sizes of the charts and the detail pages are all distinct, the funds that the user invests can be identified by comparing the sizes of the packets she receives with the publicly obtained sizes. However, inferring pie charts seems more challenging, as the entropy is much higher. This is the focus of the discussion below.

Inferring fund allocation. We first assume that the user invests in 3 funds (2-fund and 1-fund scenarios are trivial cases). The dimensions of the pie chart image are 136×136. It has 380 pixels on the circumference. Therefore the portions of two of these funds, denoted by φ and ω as indicated in Figure 10 (right), have 380 possible values each. To make the analysis easier, we conservatively adopted 0.25% as the increment to enumerate φ and ω values, which gives each fund 400 values, and totally 79401 possible charts. We observed that the sizes of the charts varied, but within a range smaller than 200 bytes. The density is over 385. At the first glance, the inference seems impossible.

Interestingly, since OnlineInvest[A] updates the pie chart every day after the market closes, the pie chart's evolution can be viewed as state transitions over a multiple-day period, initiated by the input of the first-day's financial data from the backend database and driven by the follow-up daily inputs from the market. This gives the attacker a significant reduction power, similar to that in the auto-suggestion scenario, with only two unessential differences: (1) the input comes from the backend database, not from the user; (2) the state transitions in the auto-suggestion widget are in the form of consecutive keystroke inputs, while the pie chart of each day is the evolution result of the chart of the previous day, by applying the mutual fund price changes of the current day. Since the price of each fund, in U.S. cents, is public knowledge, the pie charts on different days are indeed semantically correlated.

Following we describe an algorithm that performs such an inference. Its parameters include: `allSizes(φ,ω)` –

an array that keeps the image sizes for the 79401 pie charts, `price(fund,day)` – the prices of the invested mutual funds everyday, and `size(day)` – the image size of the pie chart of each day observed by the attacker. The program first determines the initial ambiguity set `AgtySet` based upon the image size of the first day. For each following day, the price change is applied to every pie chart in the set, and those that do not match the image size of the day are dropped. In this way, `AgtySet` shrinks everyday.

```
AgtySet= {(φ,ω)| allSizes(φ,ω) == size(1)};
for (d=2; |AgtySet| > 1; d++){ /*d is the day*/
  AgtySet'={(φ,ω)| ((φ,ω) ∈ AgtySet) ∧ ((size(d)
    == allSizes(applyDailyUpdate((φ,ω),d-1)))));
  AgtySet = {(φ,ω)| ∃(φ1,ω1) ∈ AgtySet' ∧
    ((φ,ω)== applyDailyUpdate(φ1,ω1))};
}
Output AgtySet and d;
```

We define function `applyDailyUpdate (φ1,ω1)` based on `price(fund,day)`, a function that calculates today's percentages φ and ω given yesterday's φ1 and ω1. The array `allSizes(φ,ω)` can be acquired through adjusting the money allocation in our own account to produce pie charts of different sizes. This, however, requires too much effort. A more efficient approach is to use the same GIF compression algorithm adopted by OnlineInvest^A to generate a set of the pie charts identical to those issued by OnlineInvest^A. We conducted a detailed study of some pie chart samples, and compared them with the same GIF pie charts generated by Microsoft Office Picture Manager. Although their palette scopes and color encoding are different, the lossless compression algorithm, a.k.a. Lempel-Ziv-Welch (LZW) [20], produces the same compression data in both applications. Therefore we believe that the exact GIF compression algorithm used by OnlineInvest^A can be built based on the knowledge of sample pie charts.

The experimental results. As a proof-of-concept demonstration, we used the Java package JFreeChart [10] to generate all 79401 pie charts, whose sizes fell in the range of [700,900] bytes, similar to the sizes of the sampled OnlineInvest^A pie charts. Figure 11 illustrates the relations among sizes, φ and ω for all those pie charts. The upper triangle is invalid because φ + ω > 100. The sizes of those charts form a symmetric surface. It is easy to understand the geometrical meaning of the algorithm shown earlier: given `size(d)` for a specific day, a contour line can be identified from the surface that include all the charts of that size. From this line, our algorithm drops all such points (i.e., charts) that do not evolve into the points on the next day's contour line, given the price changes of the mutual funds. The algorithm ends when there is only one point left.

Our experiments demonstrate the power of the attack. We simulated the financial market by randomly increasing or decreasing the price of each fund by [0.5%, 1%] every day. We found that typically, our algorithm successfully inferred a pie chart according to the dynamics of its sizes on 4 different days. There are some situations where the pie chart size of the fifth day is needed. Given that the initial ambiguity set contains 79401 possibilities, the result is impressive: on average, each observed pie chart size reduces the ambiguity set by more-than-one order of magnitudes.

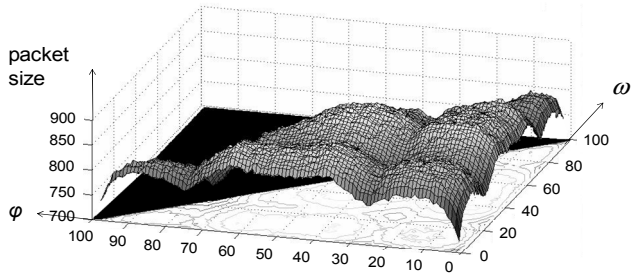

Figure 11: Pie chart sizes plotted on φ and ω axes

3) Potential More Serious Consequence

Like many financial companies, OnlineInvest^A provides a variety of account types for a broad range of customers, including investment professionals. We do not have resources to conduct the study on all these account types. However, it is possible that they also contain similar problems because it is a common practice to show price history curves and pie charts on financial pages, and these accounts probably share the same application infrastructures with the type of accounts that we studied. If an institution's or professional's account leaks out information, for example, the consequence is obviously more devastating. Of course, an institution or professional may invest in more than three funds, which increases the pie chart possibilities. On the other hand, an institution's or professional's account may be expected to be updated and viewed much more frequently. We believe that with the multiplicative reduction power, the pie chart can still be revealed.

D. Google Search, Yahoo Search and Bing Search

In addition to HTTPS extensively used to secure the web flows in web applications, cryptographic protocols for wireless communications, such as WPA and WPA2, are also found in our research to leak out a significant amount of information of the web applications that they protect. We realized that this problem is particularly serious for search engines. Although individual query words that the user enters may not be as sensitive as health, income and investment data discussed before, if an attacker can obtain her query history, the consequence can be serious: query histories reveal a lot about one's online activities, which is often viewed as sensitive information assets, particularly in corporate settings due to intellectual property concerns. To illustrate the problem, we constructed an attack program to allow an unauthorized stranger sitting outside a corporate building to glean the query words entered by employees.

1) Basics of Wi-Fi Encryption Schemes

An early scheme, WEP, is deprecated now because it is susceptible to key-recovery attacks [3]. Here we only

discuss the up-to-date Wi-Fi encryption schemes, i.e., WPA and WPA2. Using these schemes, every Wi-Fi device can establish a private channel to communicate with the wireless access point (AP). WPA is based upon TKIP, which uses RC4 stream cipher. As a result, packets encrypted by WPA exhibit byte-level granularity. WPA2 is based upon CCMP, which adopts the 128-bit AES block cipher operating in the counter mode. One might expect the "round-to-16-byte" effect due to the use of the block cipher. In fact this effect does not exist, because of the counter mode, which makes the ciphertext fully preserve the size of its plaintext, just like a stream cipher. An illustration of the CCMP packet structure is given in the IEEE 802.11 specification (see Page 180 of [9]). Therefore the reality for web developers is that they need to take the full responsibility to pad packets, as there is no mitigation at the Wi-Fi encryption layer.

2) Query Word Leaks

As discussed before, the auto-suggestion feature is vulnerable to side-channel analysis when the traffic it generates is protected by HTTPS. Many major search engines, including Google, Yahoo and Bing, also implement the feature to help users enter their queries quickly and accurately. It appears to become an important means to ensure high relevance of search results, in addition to offering friendly user interactions. Although the basic idea here is similar to that described in prior sections, a number of details and new observations are worth highlighting.

Web flow vector. Once the Wi-Fi packets are encrypted, a wireless sniffer cannot see their IP addresses but MAC addresses. However, the packets associated with auto-suggestions are easy to identify by the web flow vectors. For example, when the word "list" is entered in Google, the vector of the WPA2 packets are (*b*➔, ➔910, 96➔, *b*+1➔, ➔931, 96➔, *b*+2➔, ➔995, 96➔, *b*+3➔, ➔1007, 96➔), where *b* is around 800 bytes. As we can see from the vector, every request increases its size by one byte (*b*, *b+1*, *b+2*, *b+3*) in response to each keystroke (except backspace). The response sizes 910, 931, 995, 1007 correspond to the suggestions for "l", "li", "lis" and "list".

Ambiguity set size. Unlike OnlineHealth^{A}, which only consists of thousands of items, the number of possible query words is huge. This, however, does not impair our attack, because the number of guesses is only linear to the length of the query words. Our attack generated Google queries to match their traffic with those produced by the victim's keystrokes. The number of such attempts was only 27*(query word length), with the character set {a..z, space}.

Caching effect. Google and Yahoo set a one-hour caching period. Due to the caching, a keystroke does not generate an HTTP request if the exact request was sent recently, e.g., if one queried "aa" recently, and types "ab" in the search box, there will be no request for the suggestion for "a". However, the input here is still identifiable from the sizes of the suggestions for multiple letters, e.g., "aa" and "ab" can be determines from 26×26 2-letter combinations, even if "a" does not produce any traffic. In reality, Google's caching is seldom in effect: it disappears if the user enters the next query on the current query's result page (as oppose to always entering queries on Google's homepage), because the auto-suggestion request contains the current query word, which avoids cache hits. Bing does not cache queries.

V. Challenges in Mitigating Side-channel Threats

We have demonstrated the gravity of the information leaks in today's web applications. This section analyzes the challenges in addressing these vulnerabilities in real-world scenarios. We found that mitigation of such side-channel threats is much more difficult than it appears to be, as such an effort often needs to be *application-specific*, which means that developers must first identify the vulnerabilities in individual applications, and then think of their remedies. Identifying the vulnerabilities is challenging, as suggested by the examples in Section IV: developers need to analyze the specific program structures related to state transitions, and even semantically understand how the applications are used in various administrative environments.

One may wish that there is a "universal" mitigation so that we can fix the vulnerabilities without finding them. Section V.A assumes that developers do not know where the vulnerabilities are, but use application-agnostic mitigation approaches. We show that such mitigations are unlikely to be applied in reality due to the uncertainty of their effectiveness and the significant network overhead they incur. This finding urges an in-depth rethinking of the way today's web applications are developed.

A. Evaluations of Application-Agnostic Mitigations

It is easy to conceive high-level strategies for hiding side-channel information. Padding packets, faking superfluous (noise) packets, chopping packets into fixed-size segments and merging/splitting application states are all reasonable (and well-known) strategies. However, this does not necessarily imply that effectively deploying these strategies in real application scenarios is also easy. We can draw an analogy from the buffer overrun problem here: everyone understands the high-level strategy – buffer sizes should be correctly checked. However, how to check every vulnerable buffer in every application is non-trivial, advanced technologies need to be developed and applied, such as static analyses, type-safe languages, address space layout randomization, control-flow integrity protection, etc.

Regarding the side-channel leak problem, we focus the discussion of this section on the feasibility of finding universal mitigation policies that are both effective and agnostic to individual web applications. This examination is crucial because if we have such policies, the problem as a whole can then be solved without analyzing individual applications; otherwise, the solutions inevitably require significant efforts to identify and analyze vulnerabilities in individual applications.

Some side-channel vulnerabilities studied in prior research indeed have universal mitigations. For example, Song et al suggested a simple mitigation for the SSH inter-keystroke timing issue [15], which combines the strategies of faking noise packets and merging states to hide the timing characteristics: an SSH client always sends a packet every 50-millisecond even when the user types no keystroke or multiple keystrokes. This solves the problem with negligible network overhead, while maintaining the responsiveness of SSH. Similarly, to solve the side-channel issue of variable-bit-rate based VoIP, rounding up every packet size to 128-bit (i.e., padding the packet so that its size is a multiple of 16 bytes) is already very effective [23]. Also, in [13], the authors suggest that setting video-streaming to a constant rate would significantly reduce the attacker's ability of inferring movie titles.

Web applications, on the other hand, are much more complex than SSH, VoIP and video-streaming in terms of traffic patterns and semantics. We report here our study on how practical to deploy application-agnostic polices, based on the analyses of the applications discussed in Section IV. Note that unless explicitly denoted, all the packet sizes are in bytes, not in bits, in the following discussions.

1) On the Mitigation of OnlineHealth^A Leaks

The most promising strategy that can be applied in an application-agnostic manner is padding[2]. Here we consider two typical padding policies that can expand ambiguity sets: *rounding* that rounds up the size of every packet to the nearest multiple of Δ bytes, *random padding* that appends every packet with a padding of random length in [0,Δ). For OnlineHealth^A, the effects and the overheads of rounding and random padding are essentially the same, as they both make a packet of a size x indistinguishable from those within the Δ-byte range $[\lfloor x \rfloor_\Delta, \lceil x \rceil_\Delta)$ (rounding) or [x,x+Δ) (random padding), and on average incur the same overhead Δ/2 bytes per packet. Without loss of generality, here we just describe the analysis of rounding.

Figure 12 shows our measurements of attacker's reduction power and network overhead on each different Δ value. The network overhead was measured using a scenario of a user working on one task in every one of the 17 top-level functionalities in OnlineHealth^A. The curve for "find-a-doctor" was obtained when the city name "South Bend, IN" was used. We also plotted the curves for selection of a condition/illness through manual typing or mouse-select. In the mouse-select scenario, given a Δ value, the reduction power was calculated by identifying the conditions under the alphabetic list that became indistinguishable from each other after rounding, and then merging those conditions to measure the average ambiguity-set-sizes of individual conditions. For manual input, we measured the scenario of typing 3 or 4 keystrokes and selecting an item from the suggestions. We observed that on average the size of the suggestion list for a keystroke varied in a byte range [B, B+260]. Different prefixes in the input box correspond to different B values. Let $B + \tau = \kappa\Delta$, where τ and κ are integers and $\tau \in [0, 260]$. After the rounding, the suggestion list size can be $\kappa\Delta$, $(\kappa+1)\Delta$, …, $(\kappa+\lfloor 260/\Delta \rfloor)\Delta$. The size difference gives the attacker some information about the keystroke. In our analysis, we enumerated all 260 possible values of τ to calculate the entropy reduction associated with each τ value. The calculation showed that each keystroke leaked 1.41 or 0.724 bit entropy when Δ was 128 bytes or 256 bytes respectively. Similarly, we calculated the entropy reduction of selecting an item on the suggestion list, which was found to be much less significant. We plot the manual input curves (3- and 4- keystrokes) by combining the entropy reductions caused by all user actions.

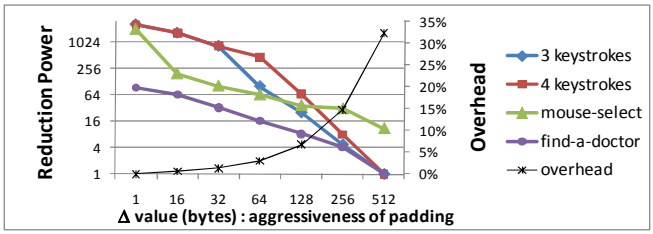

Figure 12: Reduction power and network overhead of OnlineHealth^A

Figure 12 suggests that Δ=128 is clearly unsatisfactory: the attacker's reduction powers are still 67×, 25×, 37× and 8.3× for 4-keystroke, 3-keystroke, mouse-select and find-a-doctor respectively. When Δ=256, the overhead becomes 14.8%, while the information leak then is still realistic: the reduction powers are 31× and 8× for mouse-select and 4-keystroke, which means that if the attacker suspects a person having one of 31 (or 8) possible illnesses and the person uses the mouse-select (or manual input) feature, there is a good chance that the attacker can identify the illness. This reduction power is worrisome because: (1) the set of suspected illnesses may not be very big in reality. For example, the health concern surrounding the aforementioned CEO is basically whether he is having pancreatic cancer, hormone imbalance (as the company announced) or some types of weight loss [21]; (2) the state information of the application has not been fully exploited yet in our research. Particularly, we did not utilize the probabilistic correlations among conditions, medications and procedures, due to our lack of medical knowledge: for example, a pancreatic cancer patient is more likely to receive chemotherapy than wisdom-tooth removal. Such information can be leveraged to achieve an even bigger reduction power. Finally, when Δ reaches 512 bytes, the reduction power for the mouse select scenario is reduced to 11×, and the powers for other scenarios basically vanish. The network overhead then is 32.3%. This analysis shows that application-agnostic

[2] Other common mitigation strategies, like adding noise packets, are hard to be application-agnostic for web applications. For example, fake packets unrelated to an application's behavior are identifiable by comparing the observations from two runs of the application with different inputs.

mitigation incurs a large overhead, one third of the application's bandwidth consumption, but still cannot fully subdue the information leaks.

2) On the Mitigation of OnlineTax^{A} Leaks

Figure 13 shows our evaluation of the effectiveness and the overhead of applying the rounding strategy to OnlineTax^{A}. The overhead here was calculated under the scenario where a user was working on Personal Info, Income and Credits & Deductions modules. In absence of any mitigation measures, 15 income ranges can be distinguished. Because the differences in traffic patterns are big, Δ=64 does not have any mitigation effect. When Δ=256, which significantly mitigates the leaks in OnlineHealth^{A}, the attacker can still distinguish 13 ranges. Even with Δ=1024, there are still 7 identifiable income ranges. More interestingly, increasing Δ beyond 1024 bytes does not lower the attack power, because the remaining 7 income ranges are identifiable due to the asymmetric path situation discussed earlier: a user eligible for a tax deduction needs to answer more questions than an ineligible user. The communication overhead for Δ=2048 is already 38.10%. Of course, in an extreme case, if a padding policy could make all state-transitions in the application indistinguishable, i.e., from the attacker's perspective the state diagram is just one state looping back to itself, the remaining 7 income ranges would be hidden. However, since the number and the sizes of packets involved in each state-transition varies significantly, our measurement showed that such a treatment results in a prohibitively high overhead of 21074%.

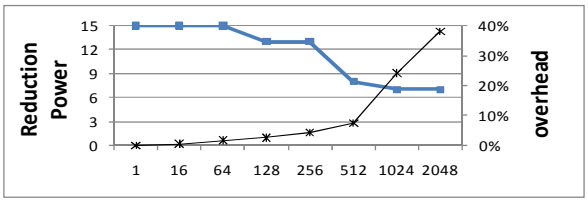

Figure 13: Reduction power and network overhead of OnlineTax^{A}

To effectively solve the problem, the application needs to merge multiple states along the longer execution paths, or produce superfluous packets to fake extra states along the shorter paths. These policies, however, are clearly application-specific, which are built upon the understanding of the state transitions within the application.

3) On the Mitigation of Search Engine Leaks

There are two reasons why the information-leaks in the high-profile search engines are hard to mitigate in an application-agnostic way: (1) these search engines need to handle high-volume search traffic, which makes the cost of applying any universal mitigations to the entire traffic unsustainably high. A more realistic alternative seems to be identifying vulnerable features (such as auto-suggestion) and applying a mitigation specifically; (2) more importantly, as we discuss below, whether certain policies can be effectively enforced also depends on application scenarios.

For example, Δ=128 seems to work for the auto-suggestion feature, because the response size generated by entering one of the 26 letters usually varies in a 200-byte range. However, choice of a specific padding policy, i.e., rounding or random-padding, is still application-specific, depending on the understanding of how the application operates. For example, we found that the auto-suggestion responses of Google/Yahoo/Bing were GZip-compressed by web servers. Typically in corporate networks, web traffic needs to go through HTTP proxies which inspect the contents to enforce security and compliance rules. The inspection requires the traffic to be decompressed. In other networks (e.g., university or home networks), there are typically no inspection proxies. As a result, the "round-up-to-Δ" policy performed on the server side becomes difficult in simultaneously hiding the packet differences in both types of environments. Thus the differences will be preserved in the Wi-Fi traffic. Other related considerations include (1) when to pad, before or after the compression, and (2) where to pad, to the HTTP header (not subject to compression) or to the HTTP body (subject to compression). For the search engines, we feel that random-padding could be a suitable mitigation if Δ is reasonably big, which may be able to ensure that the sizes of both the uncompressed packet and the compressed packet have sufficient randomness. Of course, the exact effectiveness needs more thorough evaluations.

This example, again, demonstrates the need for application-specific mitigations – even after the effective Δ value is decided, selection of the right padding policy and the practical way to enforce the policy requires application-specific information. This type of consideration was also found to be crucial for mitigating the information leaks in OnlineInvest^{A}, which we elaborate below.

4) On the Mitigation of OnlineInvest^{A} Leaks

OnlineInvest^{A} is an example to show that even for a single application, developers need to consider different factors to find effective policies. Such considerations can only be made based on the assumption that the vulnerable application features are identified.

Hiding the side-channel information of the pie chart does not require too aggressive padding. Actually, Δ=32 can already defeat a side-channel analysis, as it increases the ambiguity sets, which are already quite large, by 32 times. Alternatively, a 256-bit block cipher can be used.

However, the other two leak problems, i.e., the price-history charts and the mutual fund details pages, cannot be addressed without applying aggressive padding. Figure 14 shows that although Δ=128 may be satisfactory for the price history problem, protecting the fund detail pages requires Δ=1024. Once Δ=1024 has been applied application-wide, the network overhead is 18.8%.

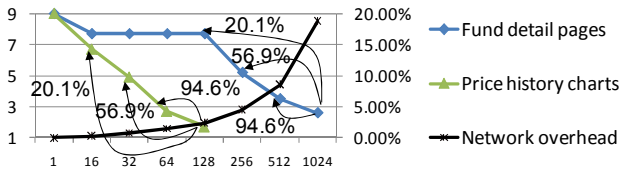

Figure 14: Account holdings anonymity, network overhead and degradations of random-padding effects upon 7-visits

More interestingly, we found that the enforceability issue, again, needs to be considered here: contrary to the search engine scenario, OnlineInvest^A should use the rounding policy, rather than random padding, because repeatedly applying a random padding policy to same responses significantly reduces its effects. Consider the price history chart problem as an example. Suppose random-[0,128)-padding has been applied several times to the same image response, making its observable sizes vary in a 100-byte range, which essentially degrades the effect of the padding to Δ=28. In our research, we ran a simulation program to calculate the probability of such degradations after applying random-padding 7 times. The program performed 2000 simulations, each generating 7 random numbers, calculating their range and subtracting it from the Δ. We observed that the padding effect was degraded to $\Delta/2$, $\Delta/4$ and $\Delta/8$ with a probabilities of 94.6%, 56.9% and 20.1% respectively when a page is visited 7 times (including the page visits by history-back and reload). Similarly, 10 visits degraded the padding effect to $\Delta/2$, $\Delta/4$ and $\Delta/8$ with probabilities of 99.0%, 74.7% and 33.9% respectively. The rounding policy is not subject to such degradations, and thus may be more suitable for protecting the OnlineInvest^A.

Finally, there is an enforcement difficulty for mashups (i.e., functionalities that import data from third-party services). Because the price history charts are fetched by the browser from the public website *https://FinancialData^A.com*, OnlineInvest^A is unable to fix the problem. They must convince FinancialData^A that because the charts are embedded in the pages of OnlineInvest^A, FinancialData^A should fix the problem, though FinancialData^A only shows public data with no confidentiality concern. Since mashups are a widely-used means for integrating third-party services, it is not realistic to require all services to pad their packets aggressively without first knowing the actual vulnerabilities.

B. *Impacts on the Application Development Practice*

The above analyses suggest that it is unlikely to have a universal remedy that fixes the whole problem without understanding the specific vulnerabilities in individual applications. Instead, the mitigation often needs to be considered as part of application development practice. Here, we summarize the technical challenges identified above and discuss the measures that need to be taken.

Challenge 1 – identifying vulnerabilities. The first technical challenge is how to find the side-channel vulnerabilities within individual web applications. As we discussed in Section III, when an application makes stateful communications and the communications are associated with user inputs or stored data, information leaks become possible. However, many applications today fit this description, and there should be more in the future. Identification of such side-channel vulnerabilities requires an in-depth analysis of the information flows within individual applications, and in some cases, acquisition of the background knowledge on how the applications are used: an example is the correlation between an illness and medicines in OnlineHealth^A. Given the trend of increasing use of web applications as substitutes for their desktop counterparts, more and more confidential user data in these applications will be subject to the side-channel threat. This urges the developer to treat the problem seriously and come up with proper testing measures to identify such weaknesses in their products during the development and testing stage.

Challenge 2 – specifying mitigation policies. Also of great importance is how to design application-specific policies that effectively and efficiently suppress the discovered information leaks. We have shown in the previous section that specifying such policies often requires nontrivial efforts to understand a web application, its traffic patterns, program structure and state transitions, and even semantic knowledge about its utilization. This makes the application developer the most suitable party for specifying those policies. As a result, an improvement is needed for the current web development practices, which calls for new technologies to be built to help the developer design and evaluate such policies.

A necessary collaborative effort – building policy enforcement infrastructures. Our study, as reported in Section V.A, indicates that enforcing well-defined policies can also be nontrivial. The enforcement often needs the collaboration among multiple parties, including the vendors of browsers (e.g., IE, Firefox) and web servers (e.g., Apache, IIS). This is because of the following observations. Unlike low-level socket programming, the design of today's web application often renders an application hard to determine the sizes of the packets it generates, as its dynamic web pages are essentially collections of element tags and macros to be expanded by server-side engines such as ASP.NET or PHP. Adding to this complexity is the encoding (usually by escaping special characters) and compression (by GZIP or DEFLATE) typically performed by web servers. This makes packet sizes even more difficult to gauge by the application developer. As a result, only the web server and the browser that work directly on the layer of network protocols, such as HTTP(S), know the exact sizes of the packets to be sent onto the network. On the other hand, the protocol layers have little ideas about the mitigation policies, as these policies often need to be application-specific, related to its state transitions. Therefore, web applications must communicate to the protocol layer about the policies. Unfortunately such a collaborative mitigation infrastructure

is unavailable on today's browsers and web servers. In Appendix C, we present a preliminary design of the infrastructure on IIS and Firefox that allows the web developers to apply well-defined padding policies.

C. *Envisioning the Future Development Process*

Although building the infrastructure for side-channel information control is a necessary step toward the solution of the problem, a more challenging task is to identify vulnerabilities and define right policies during the program development process. We envision that such a process includes at least the steps in Figure 15.

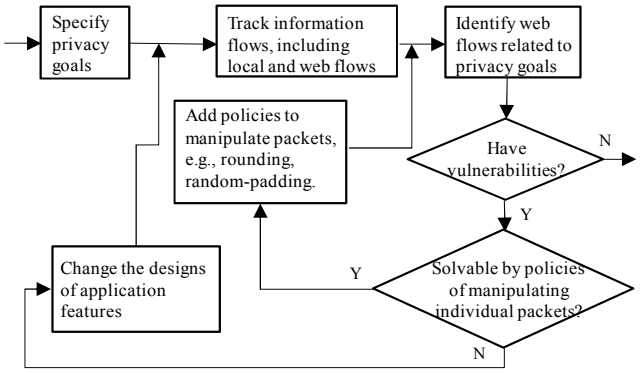

Figure 15: A development process to address side-channel threat

An application should contain clearly-specified privacy goals; static/dynamic information flow analysis needs to be applied to track the propagation of sensitive information within an application and identify violations of the goals; a traffic analysis tool is needed to ensure that sensitive web flows cannot be identified. Once a vulnerability is found, the developer needs to investigate whether it can be fixed by enforcing certain policies, such as padding or adding noise packets. If so, the right policies should be specified within the code of the application. Otherwise, the design of the application needs to be adjusted, e.g., merging program states, or re-allocating server and client side program logic, etc. OnlineTax^{A} is such an example.

Obviously, accomplishing all these tasks manually will incur too much cost. Therefore, we believe that automatic tools need to be built to assist this development process.

VI. CONCLUSIONS

A web application is split between the browser and the server, so a subset of its internal state-transitions and data exchanges inevitably go through the network. Despite encryption, some fundamental characteristics of web applications, namely *low entropy input, stateful communications*, and *significant traffic distinctions*, make the side-channel leak a realistic and serious privacy problem. As examples, we demonstrate that health records, tax information, investment secrets and search queries are being leaked out from many top-of-the-line web applications.

We also studied the challenges in mitigating such a problem, and show that effective and efficient mitigations have to be application-specific: developers will need to identify the vulnerabilities first, and then specify mitigation policies accordingly. This effort requires analysis of web application semantics, information flow and network traffic patterns. Public domain knowledge also needs to be examined in order to understand the real power of the attacker and the effectiveness of the defense.

The web industry has decisively moved into the era of software-as-a-service. Given this unquestionable context, we envision that research on disciplined web application development methodologies for controlling side-channel leaks is of great importance to protection of online privacy.

ACKNOWLEDGEMENT

We thank our colleagues at Microsoft Research: Cormac Herley offered his insights about the GIF format, and suggested the possibility of recovering a pie chart through analyzing the financial market evolution. Ranveer Chandra offered us guidance on WiFi sniffing. Emre Kiciman provided highly valuable advices on web server architecture issues, and commented on an earlier version of this paper. Johnson Apacible explained IIS implementation details. Rob Oikawa, Jim Oker and Yi-Min Wang spent significant efforts helping resolve the issues related to publishing this research. We also thank anonymous reviewers for valuable comments. Authors with IU were supported in part by the NSF Grant CNS-0716292. Rui Wang was also supported in part by the Microsoft internship program.

REFERENCES

[1] Balobardes B, Demarest S. Asking sensitive information: an example with income. Social and Preventive Medicine. Volume 48, Number 1 / March, 2003. Pages 70-72.

[2] D. Brumley and D. Boneh. "Remote timing attacks are practical," the 12th Usenix Security Symposium, 2003

[3] Andrea Bittau, Mark Handley, Joshua Lackey, "The Final Nail in WEP's Coffin," the 2006 IEEE Symposium on Security and Privacy, Oakland, CA

[4] George Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. "Privacy Vulnerabilities in Encrypted HTTP Streams." Privacy Enhancing Technologies Workshop (PET), May 2005.

[5] BusinessWeek. Privacy Survey Results. http://www.cdt.org/privacy/survey/findings/

[6] Heyning Cheng, Heyning Cheng, and Ron Avnur. Traffic analysis of ssl encrypted web browsing, 1998.

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

[8] Catalim Dima, Contantin Enea, and Radu Gramatovici. Nondeterministic nointerference and deducible information flow. Technical report TR-LACL-2006-01, LACL (Laboratory of Algorithms, Complexity and Logic), University of Paris-Est (Paris 12), 2006. http://lacl.univ-paris12.fr/Rapports/TR/TR-2006-01.pdf

[9] IEEE 802.11-2007. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Pages 180-181, http://standards.ieee.org/getieee802/download /802.11-2007.pdf

[10] JFreeChart. http://www.jfree.org/jfreechart/

[11] Renaissance Lifts Its Veil, But Just a Crack. http://www.hfalert.com/headlines.php?hid=44928

[12] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. "Hey, You, Get Off of My Cloud! Exploring Information Leakage in Third-Party Compute Clouds." ACM CCS 2009.

[13] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. "Devices That Tell On You: Privacy Trends in Consumer Ubiquitous Computing," Usenix Security, 2007.

[14] SmartAnt Telecom Co., Ltd. "Adaptor USB for Wi-Fi 802.11bg HighGain CPE," http://www.globalspec.com/FeaturedProducts /Detail/SmartAntTelecom/Adaptor_USB_for_Wi-Fi_80211bg_High Gain _CPE/97318/0

[15] Dawn Song, David Wagner, and Xuqing Tian. "Timing Analysis of Keystrokes and SSH Timing Attacks," 10th USENIX Security Symposium, 2001

[16] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata Padmanabhan, and Lili Qiu, "Statistical Identification of Encrypted Web Browsing Traffic," in IEEE Sym. on Security & Privacy 2002.

[17] Tor: anonymity online. http://www.torproject.org/

[18] Martin Vuagnoux and Sylvain Pasini. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. USENIX Security Symposium, 2009.

[19] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. The Second UNIX Workshop on Electronic Commerce, pages 29–40. USENIX Association, 1996.

[20] Wikipedia. Graphics Interchange Format. http://en.wikipedia.org/wiki/Graphics_Interchange_Format

[21] Sources about Steve Jobs' health. a) Wikipedia. Steve Jobs. http://en.wikipedia.org/wiki/Steve_Jobs; b) Steve Jobs' Health, Apple's Stock. http://www.shortnews.com/start.cfm?id=76013; c) Once Again, Apple Not Forthright About Steve Jobs' Health. http://www.businessinsider.com/2009/1/so-apple-lied-about-steve-jobs-health--again

[22] Wired News. "Declassified NSA Document Reveals the Secret History of TEMPEST," http://www.wired.com/threatlevel/2008/04/nsa-releases-se

[23] Charles Wright, Lucas Ballard, Scott Coulls, Fabian Monrose, and Gerald Masson. "Spot me if you can: recovering spoken phrases in encrypted VoIP conversations," in IEEE Symposium on Security and Privacy, May, 2008.

[24] Kehuan Zhang and XiaoFeng Wang. Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems. USENIX Security Symposium,2009

APPENDIX

A: LOW ENTROPY INPUT FEATURES IN POPULAR SCRIPT LIBRARIES

Library Name	Low entropy input feature		Library Name	Low entropy input feature
Script.aculo.us	Autocompleter		*Rico*	Ajax livegrid
Dhtmlgoodies	Ajax tooltip		*jQuery*	Ajax Event handling
OpenLaszlo	Remote Database		*TwinHelix*	AddEvent Manager
DojoCompus	Dynamic data retrieving		*Echo*	Input component with Ajax
Mochikit	Ajax sortable tables		*jsLinb*	Interact with MySQL
Crosser-Browser	Tooltips		*Rolodex*	Partial page update
Yahoo YUI	Autocomplete		*Adobe Spry*	Auto suggest

B: CIPHERS USED BY IMPORTANT HTTPS WEBSITES

Website(s)	Cipher(s)
PayPal, WellsFargo Bank, Citi Bank, Bank of America, American Express, Scottrade, E*Trade, Google Adwords, Microsoft AdCenter, eSurance, Comcast, AT&T phone account service, Provident Mortgage, GeoTrust CA, OnlineInvest[A], FinancialData[A], OnlineHealth[A], OnlineTax[A]	RC4 (a stream cipher)
Verisign CA	RC4 (stream) and AES128 (128bit block)
GEICO insurance	Triple-DES (64bit block)

C: A PRELIMINARY SIDE-CHANNEL-CONTROL INFRASTRUCTURE

Since side-channel control is a cross-layer task as discussed earlier, the very first step of such an effort, naturally, is to urge vendors of web servers and browsers to provide an infrastructure so that well-defined policies can be specified by web application developers and enforced on the protocol layer by browsers and web servers. We implemented a prototype for packet-padding as an *IIS* extension and a *Firefox* add-on, shown in Figure 16.

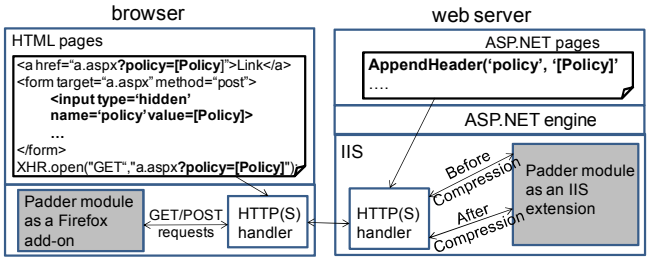

Figure 16: Padder prototypes on browser and web server

Our prototype works on the web applications written in ASP.NET. A padding policy for a response is specified by calling `AppendHeader('policy', '[Policy]')` in the corresponding ASP.NET page, where `[Policy]` is the policy definition to be discussed later. Such a policy is enforced by a padder module on the protocol layer, which intercepts the IIS workflow for generating the response and pads it according to the `policy` header. The interceptions happen before and after the HTTP compression.

For an application's browser-side component, HTTP requests come from the sources like a hyperlink, a form or an XmlHttpRequest (a.k.a. XHR) shown in Figure 16. Such a request is either GET or POST. For GET, the padding policy is inserted into the argument list in the URL, right after "?". For POST, the policy is inserted as a hidden input field. The browser-side padder is a Firefox add-on called by the HTTP handler when processing GET and POST requests.

Policy specification and enforcement. The server-side policies are specified by the following grammar. For example, a policy can be "*random-padding; 128; before-compression; header*". The policies on the browser-side are similar, except that there is no `When`-clause, as the HTTP protocol does not support compressions for requests.

```
Policy ::= Strategy ; Delta ; When ; Where
Strategy ::= rounding | random-padding
Delta ::= integer
When ::= before-compression | after-compression
Where ::= header | body
```

Enforcement of such policies is straightforward. The padder module retrieves the policy from a packet passed from the HTTP handler, calculates the packet size, applies rounding or random-padding to the packet, and then gives it back to the HTTP handler.

Functional tests. We evaluated the functionality of our prototype using a sample web application with an auto-suggestion feature implemented by XmlHttpRequest (to test AJAX-style requests), a selection list for the user to click (to test POST requests) and a search functionality similar to find-a-doctor in OnlineHealth[A] (to test server responses). We analyzed the network traffic and confirmed that the policies specified in the application were correctly enforced.