

Web-based Data Leakage Prevention

Sachiko Yoshihama¹, Takuya Mishina¹, and Tsutomu Matsumoto²

¹ IBM Research - Tokyo, Yamato, Kanagawa, Japan
{sachikoy, tmishina}@jp.ibm.com,

² Graduate School of Environment and Information Sciences,
Yokohama National University, Yokohama, Kanagawa, Japan
tsutomu@ynu.ac.jp

Abstract. The risk of data leakage from users' Web browsers to external servers has increased due to the prevalence of Software as a Service (SaaS). Such data leakage is difficult to prevent because 1) traditional firewalls cannot be used to prevent leakage without sacrificing usability, 2) coarse-grained approaches cannot distinguish data flows between trusted SaaS providers and non-trusted services, and 3) the detection of various types of sensitive data among massive amounts of Web traffic is difficult.

This paper proposes a fine-grained application-level proxy to detect potential data leakage risks. In particular, 1) it performs fine-grained analysis of the HTTP protocol and the Web content to determine the actual data flows in the massive amounts of Web traffic, 2) by analyzing the data against the traffic history, it can recognize newly generated data flows and cross-domain data flows that occur as a result of non-obvious mashups, and 3) it integrates a similarity-based content classifier to block the leakage of content of known sensitive documents, even when there are different versions.

We implemented the proposed system and evaluated its effectiveness for data extraction as well as the accuracy of classifications. The proposed system supports a holistic Data Leakage Prevention (DLP) solution that can be integrated with a document management system to identify the leakage of known sensitive data.

Keywords: Data Security, Data Leakage Prevention

1 Introduction

In Software as a Service (SaaS), application software is hosted in a service provider's infrastructure and capabilities are provided to the customers as services. The cost of the software is billed as a utility. SaaS is gaining widespread acceptance because it can reduce the initial investments and the maintenance costs. With the emergence of SaaS, Web browsers have become generic middleware for running the client-side user interface of Web-based applications, such as Web-based e-mail, online chat, file-sharing, etc. As a result, SaaS and hosted applications increase the risk of the leakage of sensitive data from a Web browser

to external servers, either because of users' mistakes or attacks by malicious adversaries.

Data leakage via Web traffic is difficult to detect or prevent for three reasons.

First, traditional perimeter defenses using firewalls cannot prevent data leakages via Web channels, because most firewalls allow HTTP connections from client machines in a private network to external servers so that internal users can use various external Web services. In addition, the amount of data flows in Web traffic tends to be too large so it is often not realistic analyze the sensitivity of data in all of the outgoing Web messages.

Second, it is difficult to distinguish between data flows to trusted SaaS providers and to non-trusted services by using a coarse-grained content-inspection approach that does not consider origins of data and destinations of data flows. In many cases, a user or a company makes a contract with a trusted SaaS provider. This means some sensitive data can be sent to the trusted SaaS provider but not to non-trusted servers. The growing trend of mashups and service integration makes the situation more complicated, because even if a user trusts a service, the trusted service may integrate other third-party services that the user does not trust. We want to detect the risks of data leaks from a trusted service to these third-party services.

Third, most of existing Data Leakage Prevention (DLP) technologies today focus on detecting specific types of sensitive data such as Personally Identifiable Information (PII) or credit-card numbers, typically by using pattern matching and dictionaries matching. However, the types of sensitive data handled by knowledge workers are more diverse, such as business secrets or intellectual properties. It is often difficult or impossible to recognize such diverse classes of sensitive data. In addition, it is difficult to identify sensitive data due to the massive amounts of data exchanged over HTTP.

This paper proposes a fine-grained application-level proxy to detect potential data leakage issues. In particular, the proposed mechanism addresses three issues above with the following approaches.

- The proposed system performs fine-grained analysis of the HTTP protocol and the Web content, such as HTML, JavaScript, XML and JSON, in order to determine the data flows in the Web traffic. In particular, it extracts the data elements from inbound and outbound Web traffic, and determines the actual data flows by comparing them with the traffic history. The method enables identification of potentially dangerous data flows among the massive amounts of communicated messages between the client-side JavaScript and the Web server.
- By recording the history of the Web traffic at the granularity of the data elements in association with their origins, the proposed system can detect data flows from one domain to another. This allows detecting cross-domain data flows that occur in client-side mashups, even though the user is not aware of such mashups.
- The proposed system can integrate any content classifiers to detect the sensitivity of the data flows on-the-fly. In this paper, we describe a classifier that

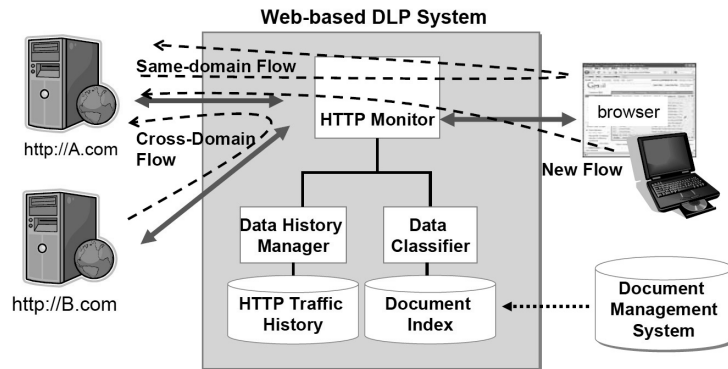


Fig. 1. Architecture of the Web-based DLP System

can be integrated with a document management system to help identify the leakage of known sensitive data. The proposed method measures the similarity of documents in a way that is robust against changes made by the users. For example, when some data is copied from a known sensitive document, that indicates a potential of high-risk of data leakage.

The rest of this paper is organized as follows. Section 2 describes the design principles and the architecture of the proposed system. Section 3 covers the prototype implementation. Section 4 presents the result of our evaluation. Section 5 discusses the related work and the contributions of this paper. Section 6 concludes the paper and discusses our future research agenda.

2 Architecture

The overall architecture of the proposed system is shown in Figure 1. The architecture consists of three major components: *HTTP Monitor*, *Data History Manager*, and *Data Classifier*.

2.1 HTTP Monitor

The HTTP Monitor intercepts HTTP requests and responses to analyze the data in the Web traffic. The HTTP Monitor recognizes the two directions of data flow, *inbound* and *outbound*. The goal of the HTTP monitor is to prevent any leakage of sensitive data from the internal domain to external domains. In other words, preventing dangerous data leakages in the outbound data flow is the objective of the proposed system. In this paper we focus on the scenario to prevent sensitive data leakage from an internal client to external servers. Therefore, in the rest of this paper, the HTTP requests sent by a client correspond to the outbound

flows while the HTTP responses sent by the servers correspond to the inbound data flows.

The HTTP Monitor analyzes the HTTP protocol to extract the data that is transferred through it. For example, the request URLs, some client-controllable HTTP header values (such as Cookie headers), and the request and response bodies convey data. Each piece of data is analyzed according to its content type. For example, each request URL is parsed, and then the key-value pairs of the request parameters are extracted from it. Likewise, the HTTP Headers are parsed and the values (such as the cookie values) are extracted.

In addition, the HTTP Monitor analyzes the content that is conveyed in the HTTP request and response body, and then parses the content recursively until each piece of content is broken down into a set of atomic data elements. This is an effective method to extract the elements of the data from Web content that is often a mix-typed. For example, HTML files typically include JavaScript embedded in the `<script>...</script>` tags. Each piece of JavaScript code is parsed to extract data elements, such as constants, string literals, and object property names in the script. Also, a string literal in JavaScript code may include HTML fragments that will later be inserted into the document DOM tree by the JavaScript code. A similar recursive analysis is done for each of the standard Web-based content types, such as HTML, JavaScript, XML and JSON. Other content types (such as office documents in a file-uploading HTTP POST request) are extracted as chunks and saved for later classification.

2.2 Data History Manager

The Data History Manager records and compares the inbound and outbound data flows at the granularity of the data elements extracted by the HTTP Monitor, determining the actual data flows, and more importantly, detecting the dangerous flows.

The data flows are categorized into three types by the Data History Manager:

Same-Domain Flows. In many cases, the client-side Web application extracts data in the inbound content, and sends it back to the server from which the content was received. (E.g., data flow from A.com to A.com via browser in Figure 1). A traditional example of this class of flow occurs with a static link in an HTML document, as well as an HTML form with hidden fields or default values. Modern Ajax applications often process data using the client-side JavaScript code that sends the data back to the server as asynchronous HTTP communications. Such same-domain flows are considered to be safe, because there is actually no real information flow from the client to the server.

Cross-Domain Flows. When some data included in the inbound content from one server is sent to another server via the client-side JavaScript code, this is considered to be a cross-domain data flow. (E.g., data flow from B.com to A.com via browser in Figure 1). The cross-domain flows create potential cross-domain data leakage risks, because these flows might be the result of

Table 1. Example of Data Flows

Data Received	from A.com (text/html)	<script> var x="abc def";</script> <input name="ghi" value="jkl">
	from B.com (text/json)	{"X": 123, "Y": 456}
HTTP Traffic History	Received from A.com	"x", "abc def", "ghi", "jkl"
	Received from B.com	"X", 123, "Y", 456
Data Being Sent	Original HTTP Request URL	http://A.com/?ghi=jkl&X=123&Z=456&msg=hello+world
	Same-Domain Flow	"ghi", "jkl"
	Cross-Domain Flow	"X", 123, 456
	New Flow	"msg", "hello world"

malicious intent, such as a malicious component in a mashup application or a cross-site scripting attack, which steals sensitive data from a trusted Web application and sends it to the attacker's server. However, due to the prevalence of mashups, there are many legitimate cross-domain flows in many of today's Web 2.0 applications.

New Flows. When some data is sent to a server and the data has not been observed in previous inbound flows, it is considered to be a new data flow. (E.g., data flow from the browser to A.com in Figure 1). New data flows may occur either as a result of user input or as a result of the execution of the client-side JavaScript code. For example, JavaScript code may generate a random value and send them to the server as a nonce.

An example of data flows is shown in Table 1.

Since the same-domain flows have low risks, the cross-domain flows and new flows are the targets of the proposed system to further assess the degree of data sensitivity by using content analysis technologies.

In addition, if the same data element is sent repeatedly, the Data History Manager ignores the redundant data elements. This design choice was made by observing the behavior of several Web applications. Our initial expectation was that the most of the new flows would be caused by user input data. However, the reality in many Ajax applications is that the majority of new data flows include values that are generated by the client-side JavaScript, such as by using random number generation or string concatenation, that represent some unique identifiers (such as session or object identifiers) or fixed keywords (such as object property names). Since such data tends to be sent repeatedly, ignoring the redundant data helps reduce the volume of the extracted data flows.

2.3 Data Classifier

The Data Classifier analyzes the data elements, and determines their sensitivities. The proposed can use any content classifiers on-the-fly to detect the sensitivity of the data flows, but in this paper we focus on a similarity-based classifier that uses data similarity as the metric of document sensitivity.

Similarity-based Classifier A key observation behind the similarity-based classifier is that knowledge workers often reuse documents and their contents to make a new document. For example, when preparing for a presentation, many people start by collecting existing documents or past presentation slides as reusable content, copy and paste the old content into the new presentation slides, and then modify them or add new content. The reused documents may originate with other people, such as coworkers. Such document reuse often happens recursively, and content propagates within an organization, or even across multiple organizations.

When working in a project team, it is often the case that a document is exchanged between team members, and each time a new revision of the document is created by each team member who adds or modifies the content. Such data exchange and group editing is often the nature of the collaboration style of white collar workers.

One problem in document reuse is that the origin and the class of the document become unclear when the documents and content are propagated. For example, some organizations define internal rules to put a “Confidential” label on each sensitive document to indicate the document class. However, such classification labels may not be properly copied into the new documents that include content copied from a sensitive document. For example, in some presentation software, the footer and header information is lost when the document template is changed. When the document content is reused recursively, people are especially likely to lose track of the document origin, and easily become unaware of the sensitivity of the document.

We propose a DLP solution that is closely integrated with document management systems in an organization. Most organizations have some kind of document management system for sensitive documents, such as a file server or database, for documents received from customers, intellectual properties, or business secrets. Such sensitive documents are often replicated on employee PCs for study or modification, but the original sensitive documents in the repository can be used as *reference documents* to detect other revisions or even part of the documents that are under a risk of exposure.

In our proposed architecture, the Document Repository represents a document management system that tracks documents along with their associated sensitivity classes. We do not assume any protection mechanisms on the Document Repository itself, but it can be a generic repository for documents associated with their sensitivity levels. The sensitivity of these reference documents can be either defined by human users or detected by using other rule-based content analysis engines. The characteristics of reference documents are extracted

from the repository, and used by the similarity-based classifier to determine the level of sensitivity in data flows. For example, we can use a simple rule-based content analysis engines to the document repository to detect documents that are marked as either “Confidential” or “Internal Use Only” by pattern matching, and give these documents sensitivity levels such as `CONFIDENTIAL`. And then, we can use the similarity-based classifier that includes the content similar to these reference documents, and detect their sensitivity level as `CONFIDENTIAL` even if they lack appropriate marking.

3 Prototype Implementation

A prototype of the proposed system was implemented on top of WebScarab [11], an open source HTTP protocol monitor. Our HTTP Monitor is implemented as a plug-in module for WebScarab, which intercepts HTTP requests and responses, parses their data, and invokes the Data History Manager and the Data Classifier.

A prototype of the similarity-based classifier was built by using the TF-IDF algorithm [14] to compare the weighted term frequencies between data. In particular, the current implementation uses Apache Lucene [7], an open source search engine that implements TF-IDF. The new flows and cross-domain flows extracted by the Data History Manager is canonicalized (e.g., by converting URL encoding and entity references to corresponding characters), and parsed by using Lucene’s built-in StandardAnalyzer to extract terms, to query the Lucene’s search index.

In the proposed prototype, a local agent software which crawls the local file system, extracts text from office documents files, and then registers the text per each page as well as a corresponding document class with Lucene’s search index. It is straightforward to extend this approach to more centralized document management systems.

4 Evaluation

The prototype was evaluated with two metrics.

First, an objective of the proposed system is to analyze high-volume Web traffic to extract the most interesting portions of the data flows. For this metric, the system is more successful if the amount of the extracted outbound data flows (“New Flows” and “Cross-Domain Flows”) are smaller compared to the size of the overall outbound data flows as well as the actual size of the user input data. Second, the proposed system is more successful if the level of the data leakage risks associated with the data class is determined properly, based on the similarities to the known data set.

4.1 Efficiency of Data Extraction

Table 2 shows the efficiency of data extraction for three popular Web applications, Gmail, Hotmail, and Twitter. Each of these services was chosen as an

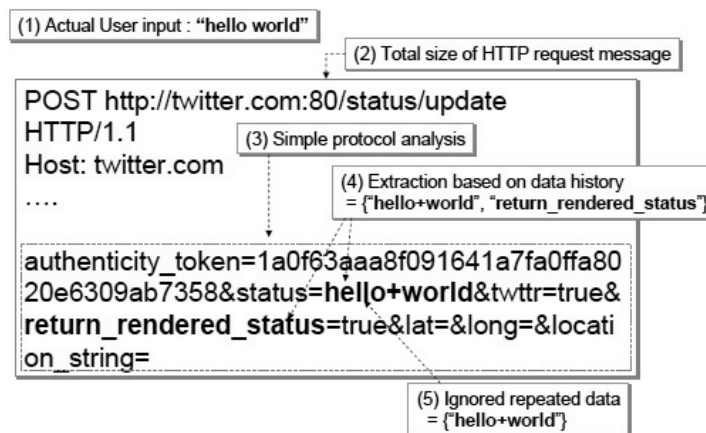


Fig. 2. Example of Data Extraction

example of what appears to be a single-domain Web application. Even though these applications perform no obvious cross-domain communication, it is expected that the proposed mechanism can detect hidden cross-domain flows if such flows exist. In the evaluation of each application, 10 user input messages were posted from Firefox browser v.3.5. Each column of the table shows the number of bytes of these types:

1. The total size of actual user input messages, such as the e-mail subject, body, and recipient address.
2. The total size of the HTTP request messages.
3. The size of the data extracted by simple protocol analysis, without using the traffic history data.
4. The size of the outbound data flow when the data is extracted based on the history. (See Section 2.2)
5. The size of outbound data flow when the repeated data flows are ignored.

An example of data extraction in Twitter's status update request is illustrated in Figure 2.

As shown in Table 2, the sizes of the extracted data can be reduced to an average of 12.63% of the data extracted by the simple protocol analysis. The extracted size is 0.7% of the total size of the HTTP request messages (Column 2 in the table). When compared with the total size of the user user input (Column 1), extracted data is about 258.92% larger than the size of the actual data. This is significantly reduced from the total HTTP request size ($663,384/1,799 = 36875\%$) or the simple protocol analysis size ($36,866/1,799 = 2049\%$).

Table 2. Evaluation : Extracted Data Sizes (in bytes)

	1. user input	2. total HTTP req	3. simple analysis	4. extract by history			5. ignore repeated			5 vs. 2	5 vs. 3	5 vs. 1
				Cross-Dom	New-Flow	Total	Cross-Dom	New-Flow	Total			
Gmail	819	132,666	4758	2	1845	1847	2	960	962	0.73%	20.22%	117.46%
Hotmail	869	490,513	25744	4409	18995	23404	380	2545	2925	0.60%	11.36%	336.59%
Twitter	111	40,205	6364	756	3812	4568	91	680	771	1.92%	12.12%	694.59%
Total	1,799	663,384	36,866	5,167	24,652	29,819	473	4,185	4,658	0.70%	12.63%	258.92%

4.2 Observations on the Cross-Domain Flows

The initial design decision about the cross-domain flow detection was made under an assumption that each Web application is built on a single Web server entry point even if the service is load-balanced at the back-end. However, while testing the prototype system, we encountered many example applications that do not satisfy this assumption.

Obviously, some of the Web applications consists of multiple Web servers interacting with the Web browsers, but which are not results of mashups, but due to the distributed architecture of the Web application. For example, when a Twitter user’s Web browser accesses `http://twitter.com/`, some resources such as JavaScript files and images are downloaded from other servers such as `http://a0.twimg.com/twitter.js`, where `a0` can be different for each access. Although this is a legitimate part of the Web application, it is detected as a cross-domain data flow when a string literal defined in `twitter.js` is sent to `http://twitter.com/`, because they belong to different domains. In addition, Twitter appears to incorporate Google Analytics to analyze the Web usage patterns, which generates quite a bit of cross-domain flows from `twitter.com` to `google-analytics.com`.

For example, we observed the following cross-domain flows in the test to post 10 status messages to Twitter:

- When loading the top page of Twitter, the main HTML file at `http://twitter.com/` imports stylesheet and JavaScript files. In each of the corresponding HTTP request, a 10 digit identifier is sent from `twitter.com` to `a*.twimg.com` where `*` is a single digit number. These identifiers are embedded in the HTML file as part of URLs.
- Some data, such as the domain name, the path, the page title, and a fixed string (“UA-30775-6”), is repeatedly sent from `twitter.com` to `www.google-analytics.com`.
- A fixed string “return_rendered_status”, which is defined in a JavaScript file in one of `a*.twimg.com`, is repeatedly sent to `twitter.com` as the name of a request parameter.
- There are a couple of other string defined in a JavaScript file in `a*.twimg.com` and sent to `twitter.com` only once each.

Since many of the cross-domain flow is actually sending the same data repeatedly, we could reduce the size of data by ignoring repeated flows. For example, the size of cross-domain data flow extracted by using traffic history (Cross-Dom of “4. extracted by history” in Table 2 is 756 bytes. But the size was reduced to 91 bytes (Cross-Dom of “5. ignore repated” in Table 2) after ignoring repeated flows. (I.e., the size in 5. is approximately 12% of 4.)

Similarly, Hotmail uses several different servers that do not even belong to the same domain. Although the server names may change dynamically, there are some clear patterns in the servers that make up each Web application. The data extraction should be more effective if we treat each set of related servers as belonging to the same domain. But even without defining such server relations, we could reduce the size of data flow from 4,409 bytes to 380 bytes by ignoring the repeated data flows. (I.e., the size in 5. is approximately 8.6% of 4.)

There was only one piece of cross-domain data flow in Gmail. In this case, a 2-letter string was sent from `mail.google.com` to `www.google.com` as the name of a request parameter.

4.3 Accuracy of the Similarity-based Classifier

We evaluated the accuracy of the similarity-based classifier in two ways. First, the stand-alone accuracy was measured by classifying the text data extracted from the test document set. Second, we built a simulated environment in which the classifier is integrated with the HTTP Monitor, the Data History Manager, and the Data Repository to evaluate the overall accuracy of the integrated DLP solution.³

Accuracy as a Standalone Classifier In order to determine the accuracy of the similarity-based classifier itself, we use three sets of office documents from three independent real-life projects, and then performed 10-fold cross-validation test with them.⁴

In the setup phase of the each round of test, the text data extracted from the documents in the test data sets are registered with the document repository along with the project name as a document class. In the test phase, we randomly chose some parts of the content from the test data sets from the three projects, and classify the content by using the similarity-based classifier. The effectiveness of the proposed method was evaluated by the *precision* and *recall*. The precision is defined as $TP/(TP+FP)$, while the recall was defined as $TP/(TP+FN)$, where TP , FP , and FN represents true-positive, false-positive, and false-negative,

³ Note that the prototype supports end-to-end behavior from the HTTP monitor to classification, but the following evaluation was done in the simulated environment, because some real-life Web applications do not allow automatic posting of many messages and try to verify a human presence using CAPTCHA.

⁴ In 10-fold cross validation, the data set from each project was split into 10 sets. In each round, 9 sets are used as the *training data sets* and the other set is used as the *test data set*. The test was repeated 10 times with a different test data set each test.

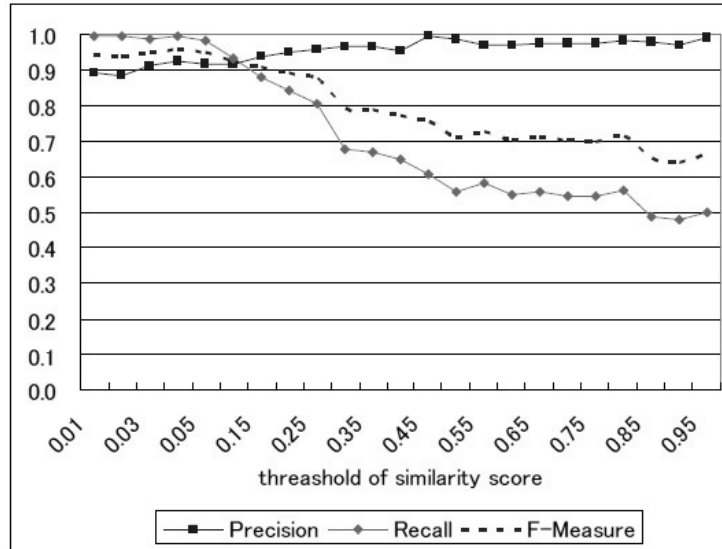


Fig. 3. Classifier Accuracy: Standalone

respectively. The F-measure was defined as $(2 * precision * recall) / (precision + recall)$. Intuitively, given a document class, the precision represents the ratio of the correctly classified pieces of data among the data classified in that given class, and the recall represents how many pieces of the data that actually belong to the class were classified as that class. The F-measure is the harmonic-mean of the precision and the recall, taking both metrics into account.

The cross-validation test was repeated by changing the threshold of the similarity score, and the results are shown in Figure 3. The highest F-Measure, 0.9601 occurs when the threshold is 0.04, for which case the precision is 0.9264 and the recall is 0.9964. Based on these results, the accuracy of the similarity-based classifier is quite good in the stand-alone environment.

Accuracy of the Integrated DLP System In the second test we simulated the integrated DLP capabilities by combining the similarity-based classifier with the HTTP Monitor and the Data History Manager, to evaluate the accuracy of the classifier over all of the extracted data. We used the same 10-fold cross validation method as in Section 4.3 and the same Web applications as in Section 4.1. In each test, text data was randomly extracted from documents in test data sets and posted to the Web application as user input message.

The result of the test (Figure 4) showed that there are quite a lot of false positive matches when classifying data elements in the cross-domain and new data flows. The false positives are mainly caused by short pieces of data, because when using the TF-IDF algorithm, the likelihood of false positives increases when the

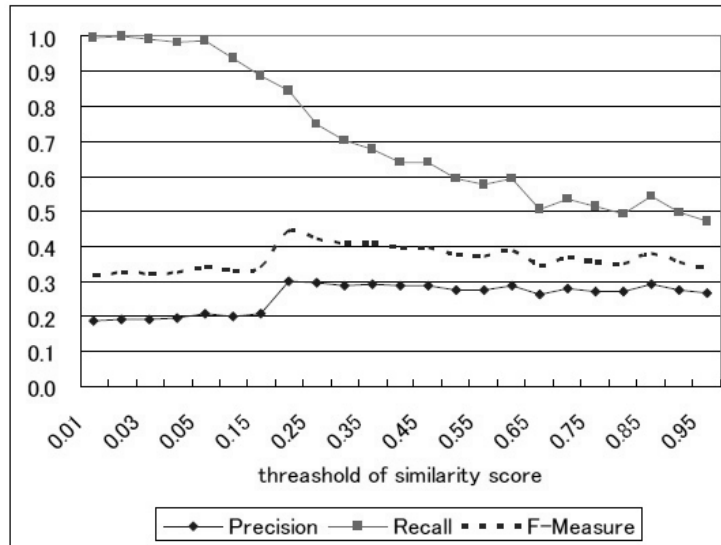


Fig. 4. Classifier Accuracy: Simulated Web Application (including short strings)

data is short. For example, when the data being sent in the outbound flow is the name of a person, then any document which includes the same name will result in a high similarity score. More prevalent examples can be found in presentation slides with words such as “backup” or “Thank you, Any questions?”. As a result, lots of short bits of data generated by the client-side JavaScript code coincidentally match the data in the document repository, causing false positives.

We modified the algorithm to ignore the short pieces of data that is less than 11 characters, because our scenario focuses on data leakage from document reuse, and it is unlikely that leakage from document reuse occur with such short strings. The improved results are shown in Figure 5. The highest F-Measure of 0.7808 was obtained when the threshold was 0.1, when the precision was 0.6599 and the recall was 0.9559.⁵

Note that in both case, the recall itself does not suffer from the presence of the short pieces of HTTP data, and thus the system can be tuned to prevent leakage with high probability at the cost of a higher false alarm rate. For example, by choosing a lower threshold such as 0.02, almost 100% of the data leakage can be prevented even though it generates about twice more false positives than true positives.

⁵ We chose 11 as the maximum length of short strings because target Web applications used in the evaluation test send a lot of generated non-confidential numeric strings with this length, which causes false positives. It is part of our future research agenda to reduce false positives without relying on heuristics about data length.

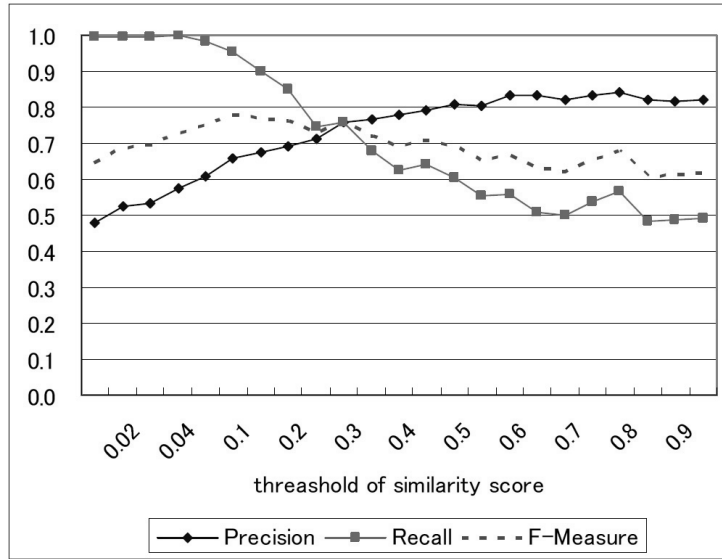


Fig. 5. Classifier Accuracy: Simulated Web Application (ignoring strings shorter than 11 chars)

5 Related Work

Some prior work that use reference monitors to track data flows between documents when they are edited by users [13][9]. However, it is often difficult to install reference monitor to all the client computers in an organization. This paper specifically focuses on the content-aware monitoring by an intermediary proxy server that removes dependencies to reference monitors in client computers. In addition, the proposed mechanism allows tracking of data between multiple client computers. For example, it can detect documents that are copied between employees operating on different computers.

Many of the content-aware DLP solutions and products today focus on network DLP, such as [1], [2], [5], and [12]. The Wi-Fi Privacy Ticker [6] searches network buffers for personal information and displays information about the exposure of sensitive terms via unencrypted channels. Some of them also provide analysis of the application-level protocols, such as filtering of the Web and instance messaging by IronPort [5] or e-mail filtering by [12]. Typical DLP solutions as well as content-analysis techniques are introduced in [10]. Due to the nature of security appliances, the detailed mechanisms of these products have not been published. However, to the best of the authors' knowledge, none of these products addresses the fine-grained data flows discussed in this paper.

Our work was inspired by the work by Borders and Prakash [3], which analyzes HTTP requests and responses to estimate the amount of the actual in-

formation flow. Our contributions include 1) extending the idea to include the concept of cross-domain flows, and 2) integrating it into a DLP solution that addresses typical data leakage problems in collaborative work environments.

There have been a lot of prior work on detecting duplicate or near duplicate documents [8]. These technologies often evolved in the area of information retrieval, trying to avoid redundant pages in search query results. The methods of similarity detection are not in the scope of this paper. We chose TF-IDF for our prototype because it is a fast and mature method.

At the same time, we observe that there are different goals for similarity (or duplicate) detection for DLP as opposed to information retrieval. In the information retrieval research, the goal is to avoid presenting duplicated pages. That means two documents should not be detected as duplicate if they are substantially different. In contrast, the goal of DLP is to detect the reuse of sensitive data, and thus two documents should be found to be similar if they shared some common text that should be protected from exposure. Therefore, in this context, TF-IDF has an advantage over traditional near duplicate document detection such as shingling [4], because TF-IDF is more robust against editorial changes and able to detect all of the documents that share common terms.

It is future research to develop improved similarity detection algorithms that address the DLP-specific requirements.

6 Conclusion and Future Agenda

This paper proposed a proxy-based system to detect and prevent data leakage via Web browsers. We implemented a prototype system and demonstrated its effectiveness through experiments with three popular applications for Web-based e-mail and micro-blogging.

However, the proposed system still has some gaps that need be filled before being used in the real-world environments.

- When SSL is used to secure the client-server communication, the proposed system can still perform content analysis when it is configured as a man-in-the-middle for SSL connections. WebScarab has an MITM mode and thus the prototype system can analyze contents transferred over SSL. However, it causes security warnings in the Web browsers due to the mismatch of the server-side certificate and the URL being accessed, and some content is not properly downloaded. This is a common problem for all of the network-based DLP.
- When a user manually encrypts a message and sends it as, for example, a message body in a Web-based mail system, the proposed classifier cannot detect the class of the content. However, the content analysis can still detect that some new data flow is occurring, and assess the amount of the information being leaked.
- The current model cannot detect covert channels by using metainformation, such as the number of data elements transmitted, the timing, or the number of non-confidential elements in an outbound data flow.

- The performance and memory consumption of the proposed system is out of the scope of this paper, but since similarity detection is implemented by using Lucene search engine, preliminary test showed that the performance overhead of the proposed method is limited and acceptable to human users. However, the implementation will need to be optimized for these aspects.

Our future research agenda include adapting some learning algorithms to learn the patterns of the data transmissions, and to ignore low-risk transmissions even if the data being sent is not identical to the past transmissions. In addition, the machine learning approach should be extended to detect covert channels.

In addition, as discussed in Section 5, the development of improved similarity detection algorithms that address DLP-specific requirements is an important challenge.

Acknowledgement

Authors wish to thank anonymous reviewers for their insightful comments and suggestions for an earlier version of this paper. Authors are also deeply grateful to colleagues at IBM Research - Tokyo and fellow students at Yokohama National University for their feedback and discussion. In particular, we appreciate Takahide Nogayama, Shannon Jacobs, Frederik De Keukelaere, and Naohiko Uramoto for reviewing and providing feedback for earlier versions of this paper.

References

1. McAfee Data Loss Prevention. <http://www.mcafee.com/japan/products/dlp.asp>.
2. RSA DLP. <http://www.rsa.com/node.aspx?id=3426>.
3. Kevin Borders and Atul Prakash. Quantifying information leaks in outbound web traffic. In *IEEE Symposium on Security & Privacy*, 2009.
4. Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *11th Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, Montreal, Canada, June 2000.
5. Cisco. Ironport.
6. Sunny Consolvo, Jaeyeon Jung, Ben Greenstein, Pauline Powledge, Gabriel Maganis, and Daniel Avrahami. The wi-fi privacy ticker: Improving awareness & control of personal information exposure on wi-fi. In *UbiComp*, September 2010.
7. The Apache Software Foundation. Apache Lucene. <http://lucene.apache.org/>.
8. J Prasanna Kumar and P Govindarajulu. Duplicate and near duplicate documents detection: A review. *European Journal of Scientific Research*, 32(4):514–527, 2009.
9. Takuya Mishina, Sachiko Yoshihama, and Michiharu Kudo. Fine-grained sticky provenance architecture for office documents. In *International conference on Advances in information and computer security*, 2007.
10. Rich Mogull. Understanding and selecting a data loss prevention solution. Technical report, Securosis, L.L.C.
11. The Open Web Application Security Project (OWASP). OWASP WebScarab Project. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.

12. Palisade Systems. Packetsure. <http://www.palisadesys.com/products/packetsure/>.
13. XiaoFeng Wang, Zhuowei Li, Jong Youl Choi, and Ninghui Li. Precip : Towards practical and retrofittable confidential information protection. In *16th Annual Network & Distributed System Security Symposium*, 2008.
14. Wikipedia. TF-IDF. <http://en.wikipedia.org/wiki/Tf-idf>.