Stochastic Modelling and Computational Sciences

PREVENTING DATA LEAKS IN WEB APPLICATIONS

V Suresh Kumar[1], Dr. Sudhamani[2] and Dr. Vankamamidi S Murthy[3]
[1]Research Scholar, Rayalaseema University, Kurnool, Andhra Pradesh, India
[2]Principal, Vivekananda Women's College, Bangalore
[3]Associate Professor, Raja Rajeswari College of Engineering,

ABSTRACT
The ubiquity of Software as a Service (SaaS) has increased the possibility of data leakage from usersWeb browsers to external servers. The inability of traditional firewalls to stop leaks without compromising usability, the inability of coarse-grained techniques to discern data flows between reliable SaaS providers and unreliable services, and the challenge of identifying different kinds of sensitive data amidst vast volumes of Web traffic make it difficult to stop such leaks.

A fine-grained application-level proxy is suggested in this paper to identify possible data leakage hazards. Specifically, it can 1) identify real data flows in the massive volumes of Web traffic by fine-grained analyzing the HTTP protocol and the content of the Web; 2) identify newly generated data flows and cross-domain data flows resulting from non-obvious mashups; and 3) integrate a similarity-based content classifier to prevent the content of known sensitive documents from leaking, even when there are different versions.

We put the suggested system into practice and assessed how well it extracted data and how accurate the classifications were. In order to detect the leakage of known sensitive data, the suggested approach facilitates a comprehensive Data Leakage Prevention (DLP) solution that may be coupled with a document management system.

Keywords: prevention of data leakage, data security

INTRODUCTION
Application software is hosted on a service provider's infrastructure and made available to clients as services under Software as a Service (SaaS) models. The programme is charged for like a utility. SaaS is becoming more and more popular since it can lower upfront and ongoing expenses. Web browsers are now general middleware for operating the client-side user interface of Web-based services, including file-sharing, online chat, Web-based email, and so on, thanks to the rise of SaaS. Therefore, SaaS and hosted applications raise the possibility that private information will leak from a Web browser to external servers due to user error or hostile adversarial attacks.

Process of Data Leakage - Limitations
For three reasons, it is challenging to identify or stop data leakage through Web traffic. Firstly, because most firewalls permit HTTP connections from client computers within a private network to external servers so that internal users can use various external Web services, traditional perimeter defencesutilising firewalls cannot prevent data leakages via Web channels. Furthermore, it is frequently impractical to assess the sensitivity of the data in every outgoing Web message due to the excessively high volume of data flows in Web traffic.

Second, employing a coarse-grained content-inspection technique that ignores data origins and destinations makes it challenging to discern between data flows to trustworthy SaaS providers and to untrusted services. A consumer or business often enters into an agreement with a reliable SaaS provider. This implies that while sensitive information can be provided to a reputable SaaS provider, it cannot be sent to unreliable servers. The scenario becomes more complex due to the growing trend of mashups and service integration, since even if a user trusts one service, it may combine other untrusted

Copyrights @ Roman Science Publications Ins.

third-party services. Our goal is to identify the potential dangers of data leaks from reliable sources to these unaffiliated businesses.

Third, the majority of Data Leakage Prevention (DLP) solutions in use today concentrate on identifying particular categories of sensitive data, such as credit card numbers or Personally Identifiable Information (PII), usually through the use of pattern matching and dictionary matching. Knowledge workers, however, manage a wider variety of sensitive data, including intellectual property and corporate secrets. It is frequently impossible or very difficult to identify such a wide variety of sensitive data classes. Additionally, because so much data is transferred over HTTP, it is challenging to distinguish sensitive data.

Gaps Identified

A fine-grained application-level proxy is suggested in this paper to identify any problems with data leaking. The following strategies are specifically used by the suggested technique to handle the three aforementioned problems.

1. To identify the data flows in the Web traffic, the suggested system carries out a fine-grained analysis of the HTTP protocol and the Web content, including HTML, JavaScript, XML, and JSON. Specifically, it takes the data pieces out of both incoming and outgoing Web traffic and uses the traffic history to compare them to identify the real data flows. The technique allows the identification of potentially harmful data flows from the enormous volume of messages that are sent between the Web server and the client-side JavaScript.
2. The suggested system can identify data flows from one domain to another by logging the history of Web traffic down to the level of individual data pieces and their origins. This makes it possible to identify cross-domain data flows in client-side mashups even when the user isn't aware of them.
3. Any content classifier can be integrated into the suggested system to instantly detect the sensitivity of the data flows. We present a classifier in this research that

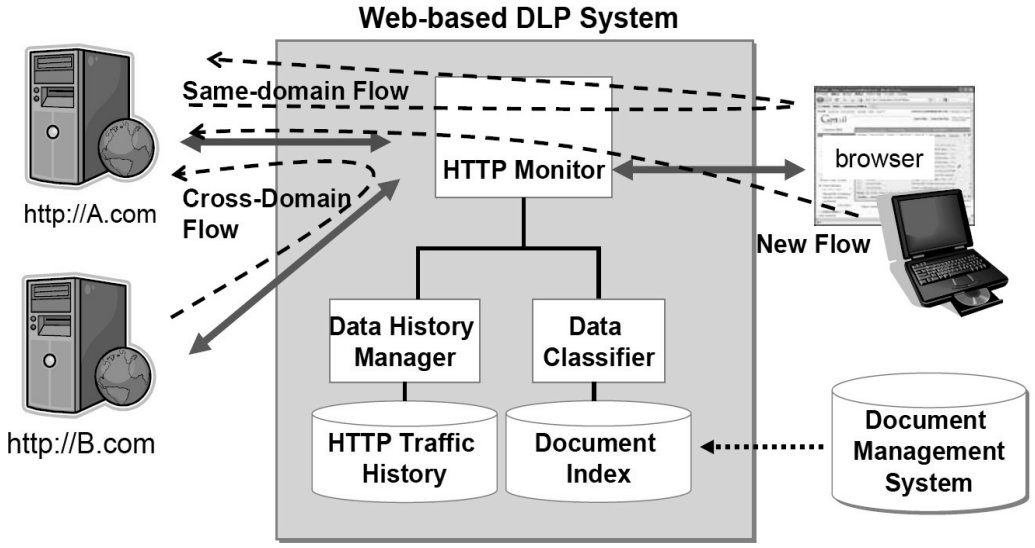

Fig.1. Web-based DLP System Architecture

can be combined with a document management system to detect known sensitive data leaks. The suggested approach calculates document similarity in a way that is resistant to user-made modifications. For instance, there may be a substantial danger of data leaking if any data is taken from a known to be sensitive document.

This is how the remainder of the paper is structured.

1. The suggested system's architecture and design principles are explained in Section 2.
2. The prototype's implementation is covered in Section 3.
3. The findings of our assessment are shown in Section 4.
4. The relevant literature and this paper's contributions are covered in Section 5. The paper is concluded with a discussion of our future research objectives

Proposed Solution

Architecture

Figure 1 depicts the general design of the suggested system. Three main parts make up the architecture: Data Classifier, Data History Manager, and HTTP Monitor.

HTTP Monitor, Data History Manager, and Data Classifier.

HTTP Monitor

The HTTP Monitor is a tool for analyzing data in Web traffic by intercepting HTTP requests and responses. Both inbound and outbound data flow directions are detected by the HTTP Monitor. Preventing sensitive data leaks from the internal domain to external domains is the aim of the HTTP monitor. Put otherwise, the goal of the suggested system is to stop harmful data leaks from occurring throughout the outgoing data flow. The scenario to stop sensitive data from leaking from an internal client to external servers is the main topic of this study. For the remainder of this work, then, the outgoing flows are represented by the HTTP requests issued by a client, and the inbound data flows are represented by the HTTP responses sent by the servers.

To extract the data delivered through the HTTP protocol, the HTTP Monitor analyses it. Data is conveyed, for instance, by the request and response bodies, the request URLs, and some client-controllable HTTP header values (such the Cookie headers). Every data point is examined in light of its particular content type. For instance, after parsing each request URL, the key-value pairs corresponding to the request parameters are extracted. Similarly, values (such cookie values) are taken from the HTTP Headers once they have been parsed.

The content that is transmitted in the HTTP request and response bodies is also examined by the HTTP Monitor, which parses the content recursively until each piece of content is divided into a collection of atomic data elements. This works well for extracting data items from web material, which is frequently mixed-typed. For instance, JavaScript is frequently included in HTML files as <script>...</script> tags. To extract data items from each JavaScript code segment, including constants, string literals, and object property names, the code is parsed. Additionally, HTML pieces that the JavaScript code will later place into the document DOM tree may be included in a string literal. Every common Web-based content type, including HTML, JavaScript, XML, and JSON, undergoes a comparable recursive analysis. Other content categories are extracted in chunks and stored for further classification (e.g., office documents in a file-uploading HTTP POST request).

Data History Manager

In order to identify the real data flows—and, more crucially, the harmful ones—the Data History Manager logs and compares the inbound and outward data flows at the level of the data items extracted by the HTTP Monitor.

The Data History Manager divides the data flows into three categories:

Same-Domain Flows: A client-side Web application retrieves data from incoming content and returns it to the server that got the material in many instances. (For instance, Figure 1 shows the data flow via browser from A.com to A.com. An HTML form with hidden fields or default values, as well as a static link in an HTML text, are classic examples of this type of flow. Data processing in modern Ajax applications frequently involves client-side JavaScript code that asynchronously communicates the processed data back to the server via HTTP. Since there isn't a true information flow from the client to the server, such same-domain transfers are regarded as secure.

Cross-Domain Flows:A cross-domain data flow occurs when certain data contained in incoming content from one server is transmitted to another server using client-side JavaScript code. (For example, Figure 1 shows the data flow via a browser from B.com to A.com. Because the cross-domain flows could be the product of

Table1. ExampleofDataFlows

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, there is a potential risk of cross-domain data leakage. Nonetheless, a lot of today's Web 2.0 apps have numerous acceptable cross-domain flows because mashups are so common.

New Flows:A data flow is deemed to be fresh when it is provided to a server and the data has not been seen in prior incoming flows. (For example, Figure 1 shows the data flow from the browser to A.com. Both user input and client-side JavaScript code execution have the potential to produce new data flows. For instance, random values may be generated by JavaScript code and sent as nonces to the server.

Table 1 provides an illustration of data flows.

The cross-domain flows and new flows are the intended system's targets in order to further evaluate the level of data sensitivity by employing content analysis technologies, as same-domain flows carry less risk.

Furthermore, the Data History Manager discards redundant data components if they are transmitted more than once. The behavior of multiple Web applications was observed to make this design decision. Initially, we assumed that user-inputted data would be the primary cause of most of the new flows. The majority of new data flows in many Ajax applications, however, are actually generated by client-side JavaScript, using techniques like string concatenation or random number generation, to represent some unique identifiers (like object or session identifiers) or fixed keywords (like object property names). Ignoring the redundant data helps lower the volume of the extracted data flows because this type of data is frequently sent.

Data Classifier

After analyzing the data items, the Data Classifier establishes their sensitivities. To identify the sensitivity of the data flows, the proposed can employ any content classifiers in real time; however, in this paper, we concentrate on a similarity-based classifier that employs data similarity as the document sensitivity metric.

Classifier Based on Similarity: A fundamental finding of the similarity-based classifier is that knowledge workers frequently create new documents by reusing old ones and their contents. For instance, a common practice while getting ready for a presentation is gathering pre-existing documents or previous presentation slides as reusable content. Then, the old content is copied and pasted into the newly created presentation slides, where it is modified or updated. The documents that are repurposed might have been created by others, such colleagues. This kind of recursive document reuse frequently occurs, and content spreads within or even between organizations.

It is common for project team members to share documents with one another, and with each exchange, a new version of the document is created by the team member who adds or changes material. Such group editing and data sharing are frequently characteristics of white collar workers' collaborative styles.

One issue with document reuse is that when content and documents are shared, it can be difficult to determine the document's original source and class. For instance, some businesses have internal policies dictating that every sensitive document must have the label "Confidential" to identify its class. These classification marks, however, might not be accurately replicated into the new papers that contain content that was taken from a private document. For instance, when the document template is altered in certain presentation software, the information in the header and footer is lost. People are especially prone to forget where a document originated and to become ignorant of its sensitivity when its material is reused often.

We suggest a DLP solution that works hand in hand with an organization's document management systems. For private or confidential information, most businesses have some sort of document management system in place, like a file server or database, for customer documents, intellectual property, or trade secrets. The original sensitive documents in the repository can be used as reference documents to identify any alterations or even portions of the documents that are at risk of exposure. These kinds of sensitive documents are frequently duplicated on employee PCs for analysis or modification.

The Document Repository in our suggested architecture is a representation of a document management system that keeps track of papers and the sensitivity classes that go with them. The Document Repository itself may serve as a general repository for documents related to their sensitivity levels, but we do not assume any protection methods on it. These reference papers' sensitivity can be determined by human users or by applying additional rule-based content analysis engines. The traits found in reference materials are extracted from the repository, and the degree of sensitivity in data flows is ascertained by the similarity-based classifier using this information. For instance, we can apply sensitivity levels like CONFIDENTIAL to documents that are designated as "Internal Use Only" or "Confidential" by pattern matching using a straightforward rule-based content analysis engine to the document repository. Then, even in the absence of the proper marking, we can identify the sensitivity level of these reference documents as CONFIDENTIAL by using the similarity-based classifier that incorporates material similar to these documents.

Prototype Implementation

An open source HTTP protocol monitor called WebScarab served as the foundation for the implementation of a proposed system prototype. Our HTTP Monitor is developed as a WebScarab plug-in module that intercepts requests and answers over HTTP, parses their content, and then launches the Data Classifier and Data History Manager. By comparing the weighted term frequencies between the data, the TF-IDF algorithm was used to construct a prototype of the similarity-based classifier. Specifically, Apache Lucene, an open-source search engine that implements TF-IDF, is used in the current implementation. In order to query Lucene's search index, the new flows and cross-domain flows that the Data History Manager collected are canonicalized (e.g., by converting URL encoding and entity references to matching characters) and parsed using the built-in Standard Analyzer in Lucene.

A local agent programme that searches the local file system, extracts text from office document files, and registers each page's text along with a corresponding document class with Lucene's search index is used in the suggested prototype. This method can be easily extended to more centralized document management systems.

Evaluation

Two Measures Were Used to Evaluate the Prototype.

First, one goal of the suggested method is to extract the most intriguing parts of the data flows by analyzing large amounts of Web traffic. When compared to the total size of outgoing data flows and the real size of user input data, the system performs better when the amount of extracted outbound data flows (also known as "New Flows" and "Cross-Domain Flows") is lower. Second, if the degree of data leakage threats related to the data class is accurately assessed, taking into account the similarities with the existing data set, the suggested system will be more successful.

Efficiency of Data Extraction

Table 2 displays the effectiveness of data extraction for Gmail, Hotmail, and Twitter, three widely used Web applications. These services were all selected to serve as

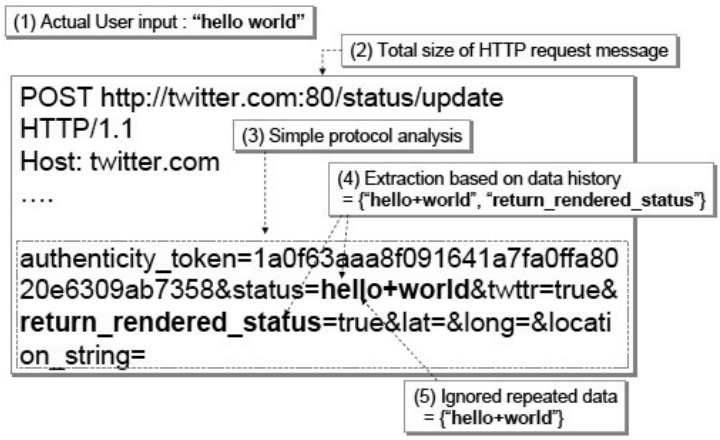

Fig.2. Example of DataExtraction

Illustrations of what seems to be a single-domain Web application. Although these applications don't engage in overt cross-domain communication, it's anticipated that the suggested mechanism will be able to find covert cross-domain flows, should they exist. Ten user-inputted messages from the Firefox browser version 3.5 were posted for each application evaluation. The table's columns each display the quantity of bytes of these kinds:

1. The entire amount of real user-inputted messages, including the body, subject, and recipient address of emails.
2. The total amount of messages in the HTTP request.
3. The amount of data that was recovered without the use of traffic history data through basic protocol analysis.
4. The amount of data that leaves the system after being extracted using historical data. Refer to Section 2.2.
5. The amount of data flow that leaves when repeated data flows are not taken into account.

Figure 2 illustrates an instance of data extraction from a status update request on Twitter.

Table 2 illustrates that the average size of the collected data may be decreased to 12.63% of the data extracted by the use of a basic protocol analysis. According to Column 2 of the data, the extracted size represents 0.7% of the overall size of the HTTP request messages. The extracted data is roughly 258.92% greater than the actual data when compared to the overall amount of the user input (Column 1). Compared to the total size of an HTTP request (663,384/1,799 = 36875%) or the size of a simple protocol analysis (36,866/1,799 = 2049%), this is far smaller.

Table 2: Evaluation:ExtractedDataSizes(inbytes)

	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%

Observations on the Cross-Domain Flows

Even if the service is load-balanced at the back end, the initial design decision regarding the cross-domain flow detection was based on the supposition that every Web application is built on a single Web server entry point. Nevertheless, we came across numerous sample apps that do not meet this assumption when testing the prototype system.

It is evident that certain Web applications involve several Web servers communicating with web browsers; nonetheless, these instances do not arise from mashups, but rather stem from the distributed architecture of the Web application. For instance, certain resources, such JavaScript files and photos, are downloaded from other sites, like http://a0.twimg.com/twitter.js, when a Twitter user's Web browser views http://twitter.com/. The value of a0 may vary depending on the request. Even if this is a valid component of the Web application, the fact that a string literal defined in twitter.js is delivered to http://twitter.com/ indicates that there is a cross-domain data flow between the two domains. Furthermore, it seems that Twitter uses Google Analytics to examine Web traffic patterns, leading to a significant amount of cross-domain flows from twitter.com to google-analytics.com.

As an example, during the test to publish ten status updates on Twitter, we noticed the following cross-domain flows:

- JavaScript and stylesheet files are imported by the main HTML file at http://twitter.com/ upon loading Twitter's home page. A 10-digit identification is transmitted from twitter.com to a*.twimg.com in each corresponding HTTP request, where * is a single-digit number. These identifiers are integrated into URLs within the HTML file.

– A fixed string ("UA-30775-6"), the path, the domain name, and the page title are among the data that are frequently transferred from twitter.com to www.google-analytics.com.

– Twitter.com receives a constant request parameter named "return rendered status" that is defined in a JavaScript file in one of a*.twimg.com.

– A*.twimg.com has a few more strings that are defined in a JavaScript file and are only transmitted once to twitter.com.

We could reduce the quantity of the data by ignoring repeated flows since many cross-domain transactions are actually conveying the same data repeatedly. For instance, Table 2's Cross-Dom of "4. extracted by history" has a cross-domain data flow size of 756 bytes that was obtained using traffic history. However, after disregarding repeated flows, the size was decreased to 91 bytes (Cross-Dom of "5. ignore repeated" in Table 2). (That is, 5.'s size is roughly 12% of 4.)

In a similar vein, Hotmail makes use of several servers that aren't even under the same name. Every Web application has servers that follow certain distinct patterns, even though the names of the servers could vary on a regular basis. If we treat every group of connected servers as if they are part of the same domain, the data extraction process ought to work better. However, by ignoring the repeated data flows, we might still reduce the size of the data flow from 4,409 bytes to 380 bytes even without defining such server interactions. (That is, 5.'s size is roughly 8.6% of 4.)

In Gmail, there was just one instance of cross-domain data transfer. In this instance, the name of a request parameter was a 2-letter string that was sent from mail.google.com to www.google.com.

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.

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.

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 respectively.

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.

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.

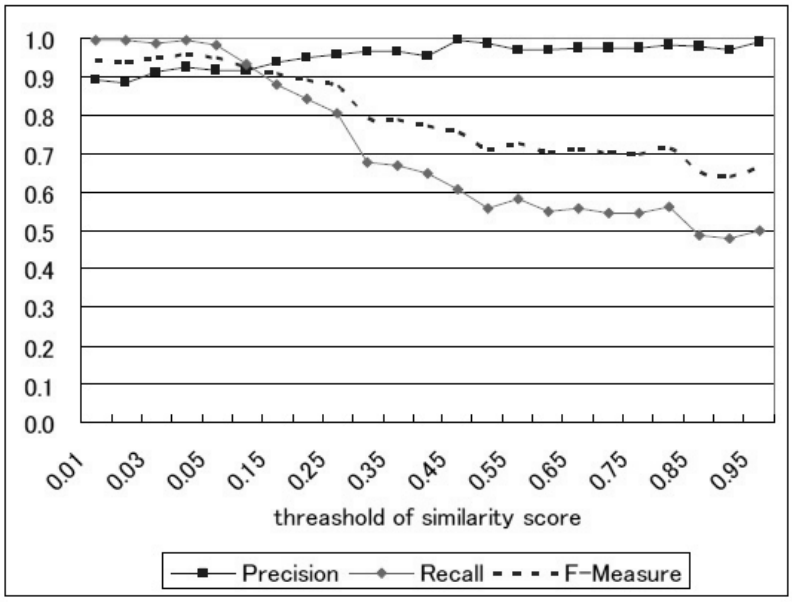

Fig. 3. ClassifierAccuracy:Standalone

These findings indicate that in a stand-alone setting, the similarity-based classifier's accuracy is fairly excellent.

The cross-validation test was repeated by changing the threashold of the similarity score, and the results are shown in Figure 3. The highest F-Measure, 0.9601 occurs when the threashold 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

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.

In order to assess the classifier's accuracy throughout the entire set of extracted data, we combined the similarity-based classifier with the HTTP Monitor and the Data History Manager in the second test, which replicated the integrated DLP capabilities. We employed the same Web apps as in Section 4.1 and the same 10-fold cross validation technique as in Section 4.3. Text data was randomly taken from test data sets' documents and uploaded as a user-input message to the Web application for each test.

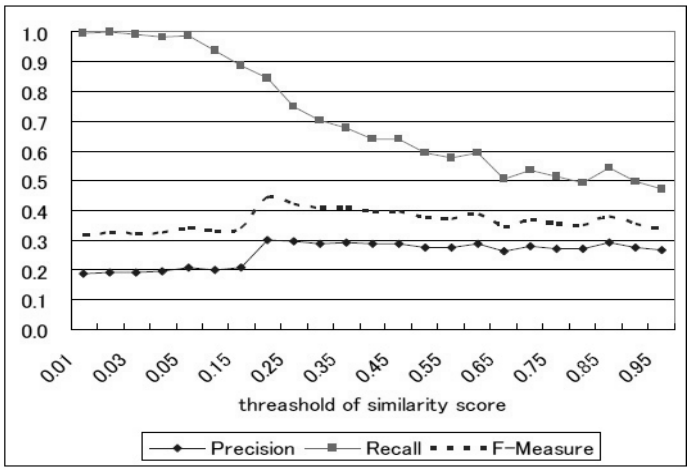

Fig.4. ClassifierAccuracy: Simulated Web Application (includingshortstrings)

When classifying data elements in the cross-domain and new data flows, there are a significant number of false positive matches, as demonstrated by the test result (Figure 4). Short data sets are the primary source of false positives because, when utilising the TF-IDF technique, the probability of false positives rises with data length. Any document with the same name will have a high similarity score, for instance, if the data being delivered in the outbound flow is a person's name. Consequently, numerous brief pieces of information produced by the client-side JavaScript code coincidentally correspond with the data present in the document repository, leading to false positives.

Because the focus of our scenario is data leakage from document reuse, we adjusted the algorithm to ignore small data strings of less than 11 characters. It is improbable that leakage from document reuse will occur with such short strings. Figure 5 displays the enhanced outcomes. When the threshold was 0.1, the precision was 0.6599, and the recall was 0.9559, the highest F-Measure of 0.7808 was recorded. 1.

It should be noted that in both cases, the presence of the brief HTTP data fragments has no negative effect on the recall itself. As a result, the system can be adjusted to very likely prevent leaks at the expense of a greater false alarm rate. For instance, nearly all data leakage can be stopped by selecting a lower threshold, like 0.02—despite producing roughly twice as many false positives as real positives.

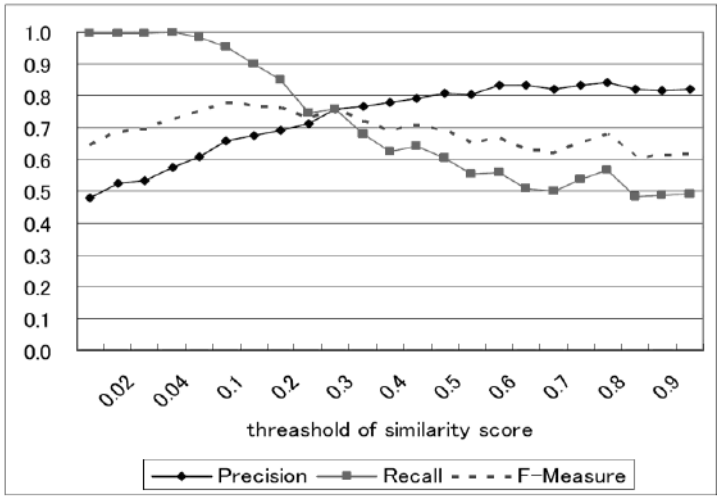

Fig.5: Classifier Accuracy: Simulated Web Application (ignoringstringsshorterthan11chars)

Related Work

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.

Installing a reference monitor on each client PC in an organisation can be challenging at times. This study focuses on the usage of an intermediary proxy server for content-aware monitoring, which eliminates the need for client computers to have reference monitors. Furthermore, the suggested method enables data tracking across several client PCs. It can identify, for instance, papers that are shared among staff members using several PCs.

Network DLP is the main emphasis of many content-aware DLP technologies and solutions available today. The Wi-Fi Privacy Ticker shows information about the disclosure of sensitive terms across unencrypted channels and scans network buffers for personal information. A few of them furthermore offer analysis of the application-level protocols, including email filtering or IronPort's Web and instance messaging filtering. Owing to the nature of security appliances, these items' intricate workings have not been made public. To the best of the writers' knowledge, none of these goods, nevertheless, handle the fine-grained data flows covered in this work.

Many research papers, which analyses HTTP requests and responses to determine the volume of the real information flow, served as inspiration for this research work. In this research paper, research contributions consist of two parts: 1) we expand the notion to encompass the idea of cross-domain flows, and 2) we incorporate it into a DLP solution that tackles common issues related to data leakage in collaborative work environments.

Many studies have already been conducted on the detection of near-duplicate or duplicate documents. These technologies have frequently developed in the field of information retrieval, with the goal of preventing duplicated pages from appearing in search query results. This work does not cover similarity detecting techniques. For our prototype, we went with TF-IDF since it's a well-established and quick method.

Simultaneously, we note that the objectives for similarity (or duplicate) identification in DLP differ from those for information retrieval. Preventing the presentation of duplicate pages is the aim of information retrieval research. That is to say, if two documents differ significantly from one another, they shouldn't be identified as duplicates. DLP, on the other hand, looks for the reuse of sensitive material, thus if two documents contain text that should be kept secret, they should be considered comparable. Because TF-IDF is more resistant to editing changes and can identify all papers that have similar phrases, it has an advantage over classic near duplicate document detection techniques like shingling in this situation.

Future work will focus on creating better similarity detection algorithms that meet the demands unique to DLP.

1 Conclusion and Future Agenda

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.

A proxy-based method to identify and stop data leaks through Web browsers was presented in this paper. We put a working prototype system into use and ran tests on three widely used Web-based e-mail and microblogging programmes to show how effective it was.

Still, there are a few holes in the suggested system that must be fixed before it can be applied in practical settings.

1. The suggested system can still carry out content analysis when SSL is used to protect client-server communication if it is set up as a man-in-the-middle for SSL connections. Because WebScarab features an MITM mode, the prototype system can examine data sent via SSL. However, because the server-side certificate and the URL being viewed don't match, certain content isn't retrieved correctly, and Web browsers issue security warnings. This is a widespread issue with all network-based DLP.
2. Upon manually encrypting a message and sending it, for instance, as the message body in an online mail system, the suggested classifier is unable to identify the content type. Nevertheless, the content analysis can still determine how much information is being stolen and identify that there is some fresh data flow.
3. By utilizing metainformation, such as the quantity of data components sent, the timing, or the number of non-confidential items in an outgoing data flow, the current model is unable to identify covert channels.

Although the performance and memory consumption of the suggested system are outside the purview of this study, early testing revealed that the suggested method's performance overhead is manageable and acceptable to humans because similarity detection is accomplished using the Lucene search engine. On the other hand, these aspects will require optimization of the implementation.

Our research objective for the future includes modifying certain learning algorithms to recognize data transmission patterns and to reject low-risk transmissions even when the data being sent differs from previous transmissions. Furthermore, it would be beneficial to expand the machine learning methodology to identify hidden channels.

Furthermore, as was covered in Section 5, there is a significant issue in creating better similarity detection algorithms that take into account DLP-specific needs.

REFERENCES

1. RSADLP.http://www.rsa.com/node.aspx?id=3426.
2. Cisco.Ironport.
3. The Apache Software Foundation. Apache Lucene. http://lucene.apache.org/.
4. The Open Web Application Security Project (OWASP). OWASP Web- Scarab Project. http://www.owasp.org/index.php/Category:OWASP WebScarab Project.
5. Wikipedia.TF-IDF.http://en.wikipedia.org/wiki/Tf-idf.