Software Engineering Institute

Insider Threat Control: Using Plagiarism Detection Algorithms to Prevent Data Exfiltration in Near Real Time

Todd Lewellen
George J. Silowash
Daniel Costa

October 2013

TECHNICAL NOTE
CMU/SEI-2013-TN-008

CERT® Division

http://www.sei.cmu.edu

Carnegie Mellon

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Department of Homeland Security or the United States Department of Defense.

DM-0000229

Table of Contents

Acknowledgments **v**

Executive Summary **vii**

Abstract **ix**

1 Introduction **1**

1.1 Audience and Structure of This Report 1

1.2 Conventions Used in This Report 1

2 Mitigating Insider Threats: Tools and Techniques **2**

2.1 The Man-in-The-Middle (MiTM) Proxy 2

2.2 The Inspection Process 2

3 Installing WebDLPIndexer **4**

3.1 Install JDK SE 6 Update 33 4

3.2 Download WebDLPIndexer 4

3.3 Install WebDLPIndexer 5

3.4 Configure and Run WebDLPIndexer 5

4 Installing GreasySpoon **7**

4.1 Download GreasySpoon 7

4.2 Install GreasySpoon 8

4.3 Configure and Run Greasyspoon 8

4.3.1 Configure GreasySpoon 9

4.3.2 Start GreasySpoon 9

4.3.3 Request Script Configuration 9

4.3.4 Add Optional Improvements 14

5 Implementation Considerations **15**

5.1 Distributed Servers for Increased Monitoring 15

5.2 Intellectual Property Repository Security 15

5.3 GreasySpoon ICAP Server Security 16

6 Use Case **17**

6.1 Actual Case 17

6.2 Intellectual Property Creation 17

6.3 Data Exfiltration Attempt 17

6.4 Detection and Response 18

6.5 Debriefing 18

Appendix **19**

References **20**

List of Figures

Figure 1: Sample Squid and ICAP Server Implementation 7

Figure 2: GreasySpoon Login 10

Figure 3: Available GreasySpoon Packages 11

Figure 4: Installing a Language Package in GreasySpoon 11

Figure 5: Installed GreasySpoon Packages 11

Figure 6: Installing a GreasySpoon Script 12

Figure 7: WebDLP Script Setup 12

Figure 8: Inserting into the WebDLP Script 13

Figure 9: Activating the WebDLP Script 13

Acknowledgments

We extend special thanks to our sponsors at the U.S. Department of Homeland Security, Office of Cybersecurity and Communications, Federal Network Resilience Division for supporting this work.

Executive Summary

One of the most pressing security challenges that organizations face today is maintaining the confidentiality of intellectual property, trade secrets, and other sensitive information. In environments that have access to the internet, the potential for data leakage is ever present.

It is reasonable to assume that a significant portion of electronically stored, sensitive information in most organizations is stored in the form of text. This text may persist in different file formats (e.g., flat text, Office Open XML, PDF), various character encodings (e.g., ASCII, UTF-8, UTF-16), and various locations within the computing infrastructure (e.g., local workstations, file servers, network traffic).

In our discussions with various organizations, it has become apparent that data loss prevention (DLP) is a difficult issue to solve in part because ideal exfiltration channels, such as modern webmail services, are readily available to insiders. In our previous technical note, *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection*, we demonstrated how an organization could prevent the exfiltration of intellectual property when it is attached to a webmail message [Silowash 2012]. However, that technical note did not address the scenario of an insider pasting text into the body of a webmail message. Therefore, a method for monitoring the content of webmail communications had to be developed.

We developed a sample solution to show how a DLP control may be implemented to monitor HTTP/S traffic for text-based data exfiltration attempts and block them in real time. By combing through the following open source software packages and libraries, a simple and elegant DLP control can be achieved:

- Apache Lucene—a powerful Java full-text indexer and search engine
- Apache Tika—a Java library for parsing text out of many different file formats
- Squid Proxy—a web proxy with the ability to decrypt SSL traffic
- GreasySpoon—a customizable ICAP server for traffic inspection and modification

The goal of the DLP control is to determine if the content in outgoing HTTP/S web requests is sufficiently similar to the organization’s corpus of intellectual property, and if so, to actively block the request and log the event.

The control has three primary components: an indexer, the GreasySpoon ICAP server, and the Squid proxy. The indexer continually builds and maintains an index of all text contained in an organization’s intellectual property files. When a client uses one of the organization’s workstations to visit a website (e.g., Gmail), the Squid proxy forwards this request to the GreasySpoon ICAP server. The ICAP server calls Java code that uses a cosine similarity algorithm—similar to algorithms used in plagiarism detection software—to search the index for bodies of text similar to the text found in the outgoing web request. If the similarity score between the web request and an intellectual property file exceeds a certain threshold, the GreasySpoon ICAP server modifies the request and hands it back to the Squid proxy in a manner that prevents the sensitive information from leaving the organization’s perimeter.

Using this control, an organization can not only protect sensitive memos, business plans, and product development specifications, but any text-based intellectual property. For organizations seeking a way to control their source code repositories, this control is an effective measure against web-based exfiltration attempts.

Abstract

In organizations with access to the internet, the potential for data leakage is ever present. Data loss prevention is a difficult issue because exfiltration channels, such as modern webmail services, are readily available to insiders. An insider can paste text into a webmail message to bypass other controls. Therefore, monitoring must include the content of this communication. A data loss prevention control determines if the content in outgoing web requests is similar to the organization's intellectual property, actively blocks suspicious requests, and logs these events. This technical note describes how a control can monitor web request traffic for text-based data exfiltration attempts and block them in real time. Using this control can help an organization protect text-based intellectual property, including source code repositories.

1 Introduction

1.1 Audience and Structure of This Report

This report is a hands-on guide for system administrators and information security teams who implement traffic-inspection technologies in their organization. We assume some experience with Ubuntu Linux and Microsoft Windows system administration in an Active Directory domain environment. This report contains detailed steps to install the necessary components so that those with little experience will have the system online quickly. The solution presented here was tested in a virtual lab with several client computers and the proxy and Internet Content Adaptation Protocol (ICAP) servers described herein.

This report extends the capabilities presented in *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012]. Therefore, we recommend to anyone implementing the methods discussed here that they review the prior publication because many of the steps presented here are reliant on a successful implementation of the Squid proxy described in that report.

It is possible to create a larger scale implementation of this solution using various methods, including the use of multiple proxies, but these methods are beyond the scope of this technical note. System administrators are encouraged to read more about the Squid caching proxy and the GreasySpoon ICAP server software to more finely tune their systems to ensure peak performance.

1.2 Conventions Used in This Report

This report contains commands that may span multiple lines. Commands are formatted using the Courier New font and end with the "↵" symbol. Each command should be entered as shown, disregarding any formatting constraints in this report. The "↵" symbol signals the end of the command and that the "Enter" key may then be pressed. An example of a multiline command in this report is

```
sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/lib/jvm/jdk1.6.0_31/bin/java" 1↵
```

2 Mitigating Insider Threats: Tools and Techniques

We define a *malicious insider* as a current or former employee, contractor, or business partner with all three of the following attributes:

- has or had authorized access to an organization’s network, system, or data
- intentionally exceeded or misused that access
- negatively affected the confidentiality, integrity, or availability of the organization’s information or information systems

Malicious insiders are able to act within an organization by taking advantage of weaknesses they find in systems. Organizations must be aware of such weaknesses and how an insider may exploit them. Organizations must also be aware of the many ways in which weaknesses are introduced.

For example, an organization may have relaxed or nonexistent acceptable-use policies for internet access. In other cases, a lack of situational awareness introduces weaknesses that malicious insiders can exploit. Additionally, an organization that allows its employees to use web-based services, such as email, increases the potential for data leakage. Establishing proper auditing policies and technical controls, as discussed in this technical note, mitigates some of these risks.

Our research has revealed that most malicious insider crimes fit into one of three categories: information technology sabotage, theft of intellectual property (IP), and fraud. This technical note focuses on the theft of information using web-based services, in particular, email accessed through a web browser using the HTTP protocol and encrypted with SSL.

The tools and techniques presented in this technical note represent only a subset of practices an organization could implement to mitigate insider threats. For example, organizations may wish to deploy commercially available software to prevent data loss. These tools and methods can be used by organizations of any size. We intentionally selected open source and public-domain tools since they are freely available to the public.

2.1 The Man-in-The-Middle (MiTM) Proxy

This report builds on the previously published technical note, *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012]. Those wishing to implement the capabilities discussed in this report will need to be familiar with the prior technical note and will need to have configured their environment accordingly. This technical note will leverage the existing work by implementing an additional ICAP server to perform additional content scanning of attachments and text entered into web pages to determine if IP is being exfiltrated.

2.2 The Inspection Process

In the prior technical note, *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection*, a Squid proxy server, C-ICAP, and ClamAV are used to inspect both clear text and encrypted web sessions for the presence of documents that may contain keywords or are tagged with hidden metadata [Silowash 2012]. Expanding on this, the CERT® Insider Threat Center developed an additional tool for inspecting documents. This tool leverages the GreasySpoon ICAP server[1] and Apache Lucene[2]. According to the Apache Software Foundation, "Apache Lucene™ is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform" [Apache 2012].

The CERT Insider Threat Center developed an agent that resides on the GreasySpoon ICAP server. This agent is written in Java and monitors a given directory of IP. When a new file is created, updated, or deleted, the agent reads the file and updates Lucene's index.

When a user uploads a document to a web-based service such as Gmail, Yahoo Mail, or Facebook, or pastes text into an online form, the document or text is compared to the indexed library of IP. If the document or text matches any of the IP above an organizationally defined threshold, the connection is blocked. Because Squid allows for the simultaneous use of multiple ICAP servers, this approach can be used with the document tagging and keyword detection capabilities discussed in the prior technical note.

[®] CERT® is a registered mark owned by Carnegie Mellon University.

[1] http://greasyspoon.sourceforge.net/index.html

[2] Software products listed in this document are for illustrative purposes; neither the Software Engineering Institute nor CERT endorses the use of the products mentioned.

3 Installing WebDLPIndexer

WebDLPIndexer is a Java agent developed by the CERT Insider Threat Center to assist with the implementation of this DLP control. It has two main functions:

1. It continuously monitors a directory of IP files and creates an Apache Lucene index based around the textual information it finds.
2. It processes query requests from any number of GreasySpoon ICAP servers to compare the index against outgoing web requests. Based on the information returned by the WebDLPIndexer, the GreasySpoon server can ultimately determine whether the outgoing request is acceptable or if it should be blocked.

WebDLPIndexer uses two additional Apache libraries aside from Lucene: Tika and Commons IO. Apache Tika is an effective Java library for parsing text from various kinds of documents. The Commons IO library was used to help continually monitor the IP directory for changes.

3.1 Install JDK SE 6 Update 33

Execute the following instructions to install JDK SE 6 Update 33:

1. To download the JDK 1.6.0_33, visit Oracle's website at the following URL:

   ```
   www.oracle.com/technetwork/java/javase/downloads/jdk6-downloads-
   1637591.html
   ```

2. Download the installer by accepting the license agreement and clicking the link for the appropriate installer.

   a. If running a 32-bit version of Linux, download the installer named

   ```
   jdk-6u33-linux-i586.bin
   ```

   b. If running a 64-bit version of Linux, download the installer named

   ```
   jdk-6u33-linux-x64.bin
   ```

3. Once downloaded, open a terminal window, change the directory to your web browser's downloads folder, and type the following:

   ```
   sudo chmod +x jdk-6u33-linux*↵
   sudo ./jdk-6u33-linux*↵
   sudo mv jdk1.6.0_33 /usr/lib/jvm/↵
   sudo update-alternatives --install "/usr/bin/java" "java"\
   "/usr/lib/jvm/jdk1.6.0_33/bin/java" 1↵
   ```

4. Select the newly installed java version by running the following command and selecting the entry for the new JDK:

   ```
   sudo update-alternatives --config java↵
   ```

3.2 Download WebDLPIndexer

To download WebDLPIndexer, execute the following instructions:

1. Change to a temporary working directory:

```
cd /tmp↵
```

2. Download the WebDLPIndexer:

```
wget http://www.cert.org/insider_threat/controls/WebDLPIndexer.zip↵
```

3.3 Install WebDLPIndexer

WebDLPIndexer now needs to be extracted and moved to an installation directory. Enter the following commands to install WebDLPIndexer:

1. Extract the archive:

```
sudo unzip WebDLPIndexer.zip↵
```

2. Create the installation directory:

   a. If WebDLPIndexer is being installed on the same server as described in the technical note *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012], or if WebDLPIndexer resides on the same server as the GreasySpoon ICAP server, then enter the following command:

   ```
   sudo mkdir /opt/indexer↵
   ```

   c. If this is a stand-alone WebDLPIndexer server, enter the following commands:

   ```
   sudo mkdir /opt↵
   sudo mkdir /opt/indexer↵
   ```

3. Move the extracted files to the installation directory:

   ```
   sudo cp -R /tmp/WebDLPIndexer/dist/* /opt/indexer/↵
   ```

3.4 Configure and Run WebDLPIndexer

To configure WebDLPIndexer, execute the following instructions:

1. WebDLPIndexer is packaged as a JAR file, which can be rudimentarily run from the command line as follows:

```
java -jar WebDLPIndexer.jar <arguments>↵
```

Currently, the agent takes five arguments with an optional sixth one. The arguments are specified in order as follows:

I. index directory—the path of the directory in which to store the Apache Lucene index

II. intellectual property directory—the path of the directory containing the IP files

III. new index (true/false)—whether or not the index should be built from scratch when the agent's execution begins

IV. keep deleted files (true/false)—whether or not files should be removed from the index if they are deleted from the intellectual property directory

V. debug mode (true/false)—whether or not debug mode should be enabled (provides verbose output)

VI. [optional] server port—the TCP port number to which the agent should listen for requests from GreasySpoon ICAP servers (if not provided, the default is 22345)

2. A bash script called *start.sh* is provided with WebDLPIndexer to ease the startup of the agent. The arguments for the script can be configured in the *config.conf* file by typing the following:

   ```
   cd /opt/indexer/↵
   sudo nano config.conf↵
   ```

   An example *config.conf* appears as follows:

   ```
   INDEX=/opt/indexer/index
   IP_DIR=/opt/property
   NEW_INDEX=true
   KEEP_DEL_FILES=false
   DEBUG=false
   PORT=22345
   ```

3. Once this file is configured, the agent can be started using the bash script as follows:

   ```
   sudo ./start.sh↵
   ```

4 Installing GreasySpoon

GreasySpoon is another ICAP server that can be used by itself or in concert with other ICAP servers. For the purposes of this technical note, we use GreasySpoon in concert with C-ICAP as discussed in the technical note *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012].

This use allows for a layered security approach whereby the methods discussed in the prior technical note can be implemented in conjunction with the methods described here. Figure 1 illustrates a possible multiple Squid, C-ICAP, and GreasySpoon implementation. The arrows show communications between the servers.

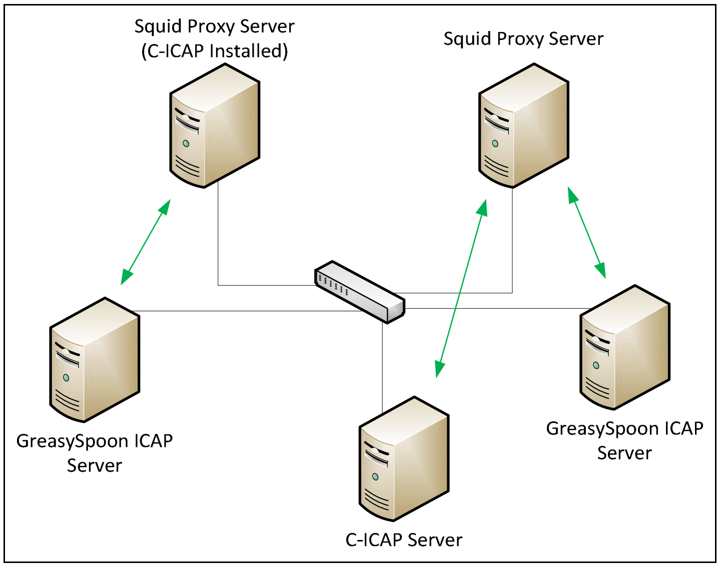


Figure 1: Sample Squid and ICAP Server Implementation

GreasySpoon allows administrators to write scripts to modify both outgoing web requests (REQ) and incoming web responses (RESP). These scripts can be written in a number of languages, including Javascript, Java, and Ruby. In our control, we use GreasySpoon to call a simple custom Java API called WebDLPClient to forward outgoing web requests to the WebDLPIndexer agent for comparison against an index of IP.

4.1 Download GreasySpoon[3]

To download GreasySpoon, execute the following instructions:

1. Change to a temporary working directory:

[3] At the time of this writing, the GreasySpoon project was still active on SourceForge. However, the official project has since been discontinued and removed from SourceForge. A mirror of the project exists on Google Code. All links have been updated to reflect this change.

```
cd /tmp↵
```

2. Download the GreasySpoon archive:

   ```
   wget https://greasyspoon.googlecode.com/files/greasyspoon-release-
   1.0.8.tar.gz↵
   ```

4.2 Install GreasySpoon

GreasySpoon now must be extracted and moved to an installation directory. Enter the following commands to install GreasySpoon:

1. Extract the archive:

   ```
   tar xvzf greasyspoon-release-1.0.8.tar.gz↵
   ```

2. Create the installation directory:

   a. If GreasySpoon is being installed on the same server as described in the technical note *Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012], then enter the following command:

      ```
      sudo mkdir /opt/greasyspoon↵
      ```

   b. If this is a stand-alone Greasyspoon server, enter the following commands:

      ```
      sudo mkdir /opt↵
      sudo mkdir /opt/greasyspoon↵
      ```

3. Move the extracted files to the installation directory:

   ```
   sudo cp -R /tmp/greasyspoon-release-1.0.8/* /opt/greasyspoon/↵
   ```

4. Download the Java extension for GreasySpoon:

   ```
   wget https://greasyspoon.googlecode.com/files/java-1.0.1.gsx  -O java-
   1.0.1.gsx↵
   ```

5. Move the file to the GreasySpoon *pkg* directory:

   ```
   sudo mv /tmp/java-1.0.1.gsx  /opt/greasyspoon/pkg/↵
   ```

6. Download *WebDLPClient*, a Java API developed by the CERT Insider Threat Center for allowing the ICAP server to communicate with WebDLPIndexer (described later in the report):

   ```
   wget http://www.cert.org/insider_threat/controls/WEBDLPClient.zip↵
   ```

7. Unzip the contents of WebDLPClient.zip:

   ```
   sudo unzip WebDLPClient.zip↵
   ```

8. Move the file to GreasySpoon’s *lib* directory. This directory is where external code libraries can be accessed and used by GreasySpoon scripts.

   ```
   sudo mv /tmp/WebDLPClient/dist/WebDLPClient.jar /opt/greasyspoon/lib/↵
   ```

4.3 Configure and Run Greasyspoon

The GreasySpoon ICAP server will likely require some basic setup configuration to integrate with your existing environment. Therefore, while the steps in Section 4.3.1 could be considered optional, the steps in Section 4.3.3 are required for the successful implementation of this control.

4.3.1 Configure GreasySpoon

To configure GreasySpoon for your environment, execute the following instructions:

1. First, GreasySpoon must be configured to use the desired version of Java. Note: For GreasySpoon to successfully execute the WebDLP script described later, a 1.6.x version of the Java JDK must be installed on the system. For testing, we used the Java SE 6 Update 33 version of JDK and installed it in the `/usr/lib/jvm/jdk1.6.0_33` directory.

    a. Edit the greasyspoon startup script:

    ```
    sudo nano /opt/greasyspoon/greasypoon↵
    ```

    b. On the line that starts with `JAVA_HOME=`, enter the path of the JDK directory as follows:

    ```
    JAVA_HOME=/usr/lib/jvm/jdk1.6.0_33
    ```

    c. Save the configuration file by pressing <CTRL> + <O> and then <ENTER>; then exit *nano* by pressing <CTRL> + <X>.

2. There are a few notable configuration parameters in GreasySpoon's *icapserver.conf* file. It can be edited using the following command:

    ```
    sudo nano /opt/greasyspoon/conf/icapserver.conf↵
    ```

3. Under the section *Define service(s) running in REQ and RESP modes…*, the following line may need to be edited to define the port on which GreasySpoon should run. By default, the port number is set to 1344; however, for the purposes of this technical note we will use port 11344 to avoid conflicts with other ICAP servers that may be on the same system:

    ```
    icap GreasySpoon * 11344 greasyspoon.ini
    ```

4. At the end of the file, a few more useful configuration parameters are present. Namely, the *admin.port* may need to be changed if you wish have a different web interface port. Additionally, it may be useful to set the *admin.ssl* parameter to *true* if you wish to encrypt your traffic to the GreasySpoon web interface. We recommend that SSL encryption be used on any login page that requires a user to enter credentials. Configuring SSL may require the installation of additional packages and additional configuration, which is beyond the scope of this technical note.

4.3.2 Start GreasySpoon

To start the GreasySpoon ICAP server, execute the following commands:

```
cd /opt/greasyspoon↵
sudo ./greasyspoon start↵
```

4.3.3 Request Script Configuration

The GreasySpoon web interface can be accessed using your preferred web browser. In our virtual environment, the name of the GreasySpoon server is *gs.corp.merit.lab*, it is running on the default port of 8088, and SSL is disabled.

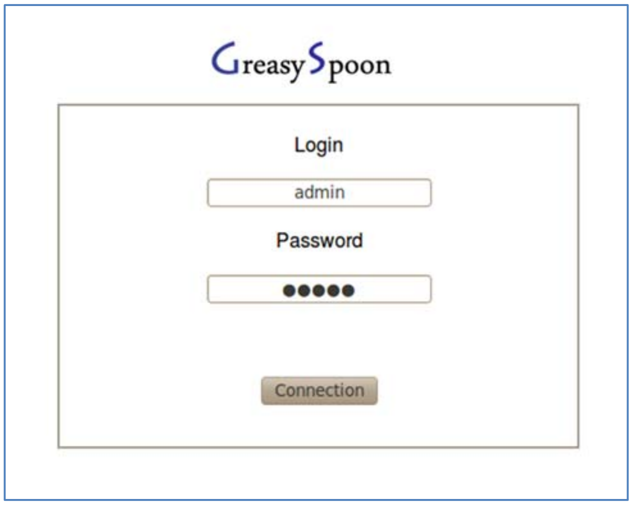
To configure the request script for the GreasySpoon ICAP server, execute the following instructions:

1. Enter the following URL into the web browser, but change the protocol (http or https), hostname, and port number based on your GreasySpoon ICAP server configuration:

```
http://gs.corp.merit.lab:8088↵
```

Note: It is also possible to access the GreasySpoon administration page by substituting the fully qualified domain name (FQDN) with the server's IP address (e.g., http://192.168.59.20:8088).

2. You should see the administrative login page. Enter *admin* as the username and *admin* as the password (i.e., the default credentials) and click *Connection*. Note: We recommend changing these default credentials as soon as possible using the GreasySpoon web interface.[4]



Login

admin

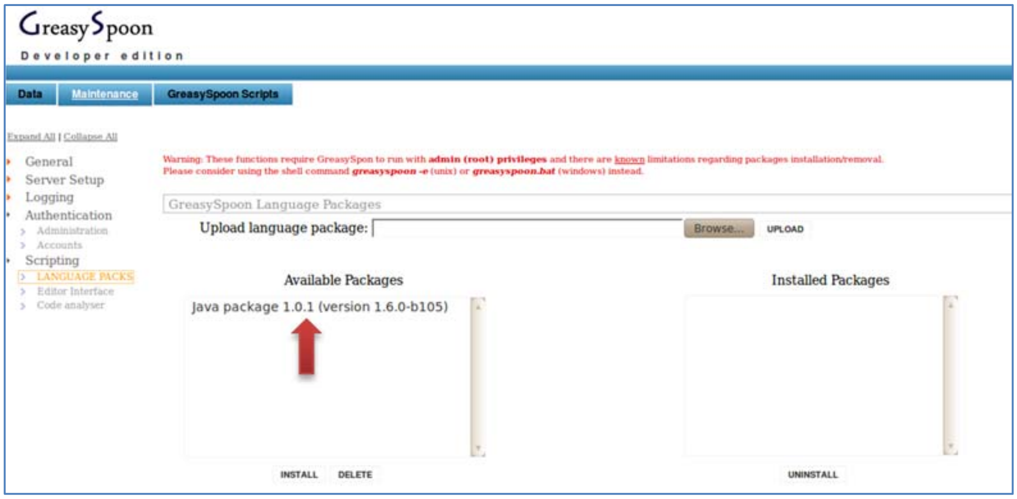
Password

●●●●●

Figure 2: GreasySpoon Login

3. Click on the *Maintenance* tab, then *Scripting*, and finally *Language Packs*.
4. On the line that reads *Upload language packs*, click *Browse* and navigate to `/opt/greasyspoon/pkg/java-1.0.1.gsx`. Once the file is selected, click *Open* to close the dialog box and then click *Upload* to submit the file. After a short time, the package should be shown under the *Available Packages* column as shown in Figure 3.

[4] See http://greasyspoon.sourceforge.net/setup_gs.html for instructions on changing the administrator password.



Warning: These functions require GreasySpon to run with **admin (root) privileges** and there are known limitations regarding packages installation/removal.
Please consider using the shell command ***greasyspoon -e*** (unix) or ***greasyspoon.bat*** (windows) instead.

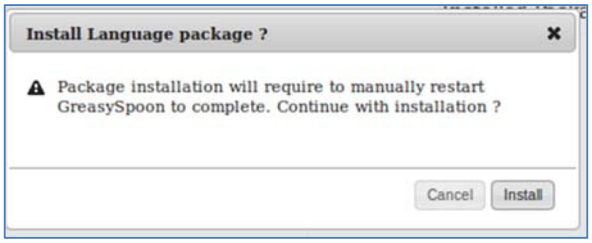
GreasySpoon Language Packages

Upload language package:

Available Packages	Installed Packages
Java package 1.0.1 (version 1.6.0-b105)	

Figure 3: Available GreasySpoon Packages

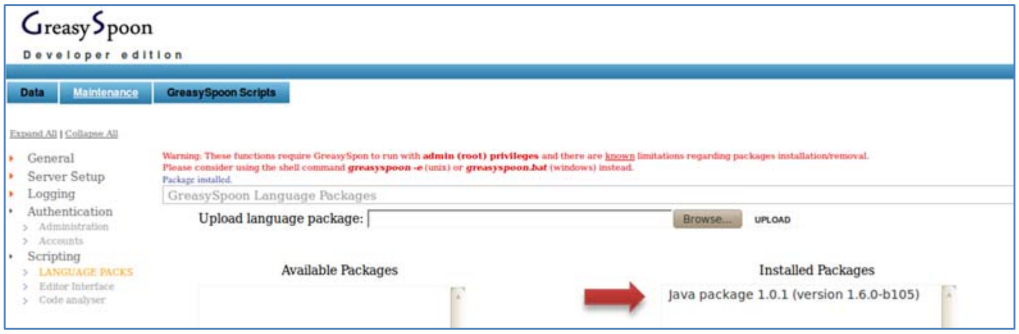
5. Click the file under *Available Packages* and click *Install*. You will be prompted to confirm the package installation as shown in Figure 4. The name and version of the package should eventually be displayed under the *Installed Packages* column as depicted in Figure 5.



Install Language package ?

Package installation will require to manually restart GreasySpoon to complete. Continue with installation ?

Figure 4: Installing a Language Package in GreasySpoon



Warning: These functions require GreasySpon to run with **admin (root) privileges** and there are known limitations regarding packages installation/removal.
Please consider using the shell command ***greasyspoon -e*** (unix) or ***greasyspoon.bat*** (windows) instead.
Package installed.

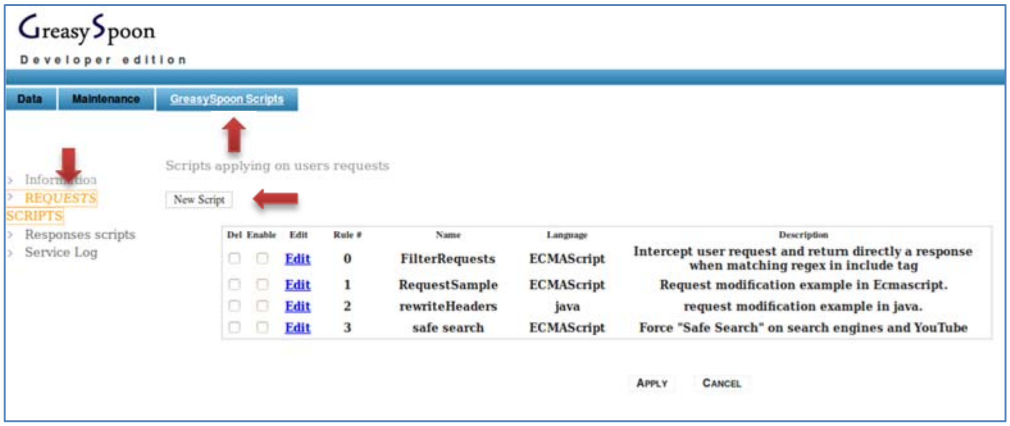
GreasySpoon Language Packages

Upload language package:

Available Packages	Installed Packages
	Java package 1.0.1 (version 1.6.0-b105)

Figure 5: Installed GreasySpoon Packages

6. Once the package is successfully uploaded, navigate to the *GreasySpoon Scripts* tab and click on *REQUEST SCRIPTS* as shown in Figure 6.

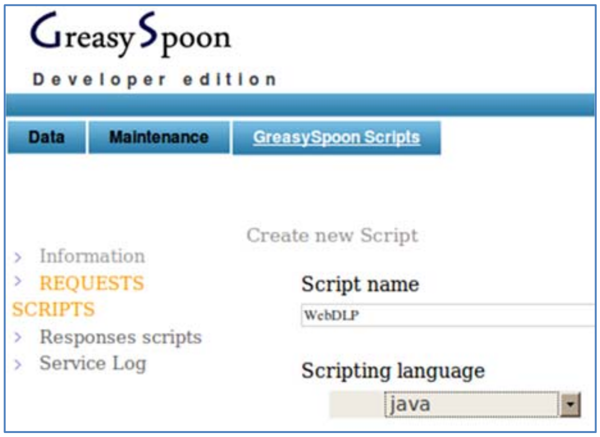


Scripts applying on users requests

Del	Enable	Edit	Rule #	Name	Language	Description
□	□	Edit	0	FilterRequests	ECMAScript	Intercept user request and return directly a response when matching regex in include tag
□	□	Edit	1	RequestSample	ECMAScript	Request modification example in Ecmascript.
□	□	Edit	2	rewriteHeaders	java	request modification example in java.
□	□	Edit	3	safe search	ECMAScript	Force "Safe Search" on search engines and YouTube

Figure 6: Installing a GreasySpoon Script

7. Click *New Script*.
8. Enter the title *WebDLP* for the script as shown in Figure 7 . Select *Java* as your scripting language of choice and click *Apply*.



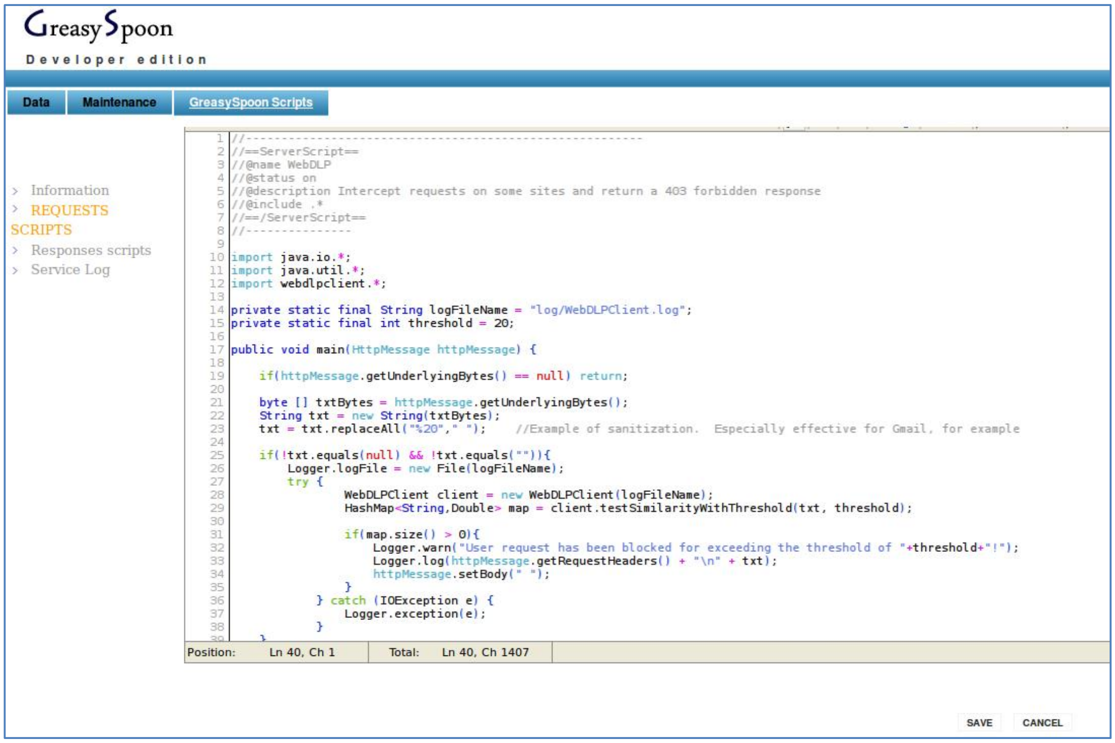
Create new Script

Script name: WebDLP

Scripting language: java

Figure 7: WebDLP Script Setup

9. Executing step 8 should bring up the script editor page. Copy the text under the *WebDLP GreasySpoon Request Script* from the Appendix into the editor window as shown in Figure 8.



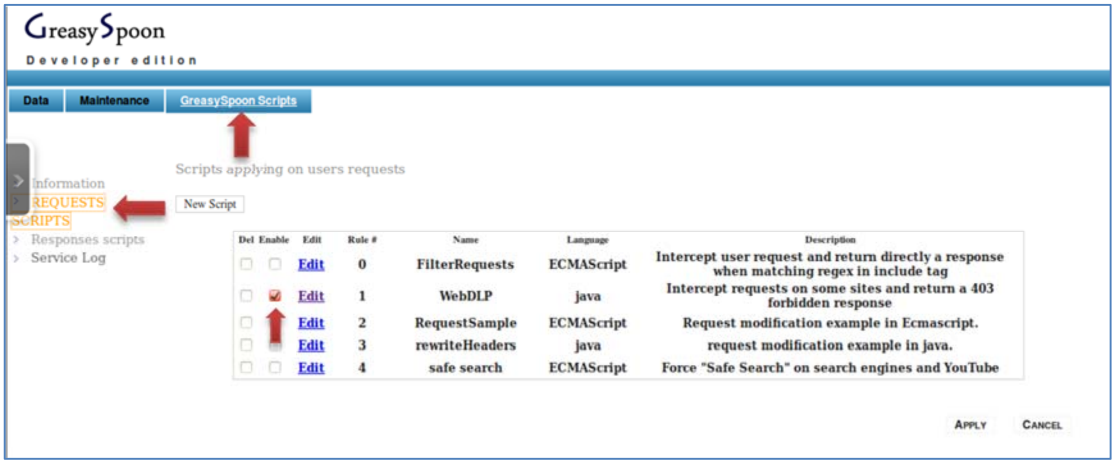
```
//---------------------------------------------------------
//==ServerScript==
//@name WebDLP
//@status on
//@description Intercept requests on some sites and return a 403 forbidden response
//@include .*
//==/ServerScript==
//---------------

import java.io.*;
import java.util.*;
import webdlpclient.*;

private static final String logFileName = "log/WebDLPClient.log";
private static final int threshold = 20;

public void main(HttpMessage httpMessage) {

    if(httpMessage.getUnderlyingBytes() == null) return;

    byte [] txtBytes = httpMessage.getUnderlyingBytes();
    String txt = new String(txtBytes);
    txt = txt.replaceAll("%20"," ");    //Example of sanitization.  Especially effective for Gmail, for example

    if(!txt.equals(null) && !txt.equals("")){
        Logger.logFile = new File(logFileName);
        try {
                WebDLPClient client = new WebDLPClient(logFileName);
                HashMap<String,Double> map = client.testSimilarityWithThreshold(txt, threshold);

                if(map.size() > 0){
                    Logger.warn("User request has been blocked for exceeding the threshold of "+threshold+"!");
                    Logger.log(httpMessage.getRequestHeaders() + "\n" + txt);
                    httpMessage.setBody(" ");
                }
            } catch (IOException e) {
                Logger.exception(e);
            }
    }
```

Position: Ln 40, Ch 1 Total: Ln 40, Ch 1407

Figure 8: Inserting into the WebDLP Script

10. Click *Save* at the bottom of the window to save the script. Click on the sidebar link titled *REQUEST SCRIPTS.*

11. Ensure that the *Enabled* check box for the WebDLP script is checked, and click *Apply* as shown in Figure 9. This step should immediately enable the GreasySpoon ICAP server to apply the script to any requests that are sent to it.



Scripts applying on users requests

Del	Enable	Edit	Rule #	Name	Language	Description
☐	☐	Edit	0	FilterRequests	ECMAScript	Intercept user request and return directly a response when matching regex in include tag
☐	☑	Edit	1	WebDLP	java	Intercept requests on some sites and return a 403 forbidden response
☐	☐	Edit	2	RequestSample	ECMAScript	Request modification example in Ecmascript.
☐	☐	Edit	3	rewriteHeaders	java	request modification example in java.
☐	☐	Edit	4	safe search	ECMAScript	Force "Safe Search" on search engines and YouTube

Figure 9: Activating the WebDLP Script

4.3.4 Add Optional Improvements

Web applications often use unique or proprietary text-based protocols for communicating between clients and servers over HTTP. For example, the body contents of an HTTP POST method in one web-based chat program will likely be different from POST methods in other web applications. One method would send the user's message without any modification whatsoever, while another would change the text's character set, add parameterized values, and pass certain hidden flags to the server.

Depending on the underlying text protocol being used by certain web application, varying thresholds may need to be set for requests from specific services. For example, in our testing we found that a threshold of 20% would catch a certain body of text passing through Yahoo! Mail but not Gmail. By narrowing the threshold to 15%, the body of text was then also caught when passed through Gmail. Our suggestion for dealing with this issue is to have separate GreasySpoon scripts (like the WebDLP script) that only inspect requests from certain URLs.

The first few lines of a GreaseSpoon script are a commented header section that GreasySpoon treats as important metadata. One of those lines contains the string *@include*, which in our WebDLP script is configured as follows:

```
//@include .*
```

This configuration directive tells GreasySpoon to apply this script to all requests that come to the GreasySpoon ICAP server. If we wanted the script to apply only to outgoing requests to Gmail, we would change the line to be the following:

```
//@include https://mail.google.com.*
```

We suggest that you have multiple, nearly identical GreasySpoon scripts that focus on dealing with requests from specific webmail applications, as in the following example:

1. Create a new script as described in Section 4.3.3, steps 7 through 11.
2. Give the script a unique title that identifies its purpose (e.g., *WebDLP-Gmail*)
3. Edit the *@include* header line to correspond to the URL desired (e.g., `http://mail.google.com.*`).
4. Change the threshold value on line 16 to your desired value.

By testing over time, organizations will be able to identify their optimal thresholds for permissible web applications. The additional benefit of providing separate scripts for each web application is that, with some time and energy, the specific application's text-based protocol could be reverse engineered enough so that the script could remove the unnecessary protocol-specific text and parse out just the user's inputted text from the request. This approach will in turn allow WebDLPIndexer to compute more accurate cosine similarity scores because it will not factor in the unnecessary, application-specific text.

5 Implementation Considerations

Organizations may wish to implement multiple servers within the organization to increase performance as illustrated in Figure 1. Multiple proxy servers may utilize one or more ICAP servers for content adaptation. For example, one ICAP server may scan content for known viruses while the other uses the methods presented in this report to ensure IP is not leaving the organization. Commercial and open source proxy servers, like Squid proxy, support multiple ICAP servers.

5.1 Distributed Servers for Increased Monitoring

Organizations can use a distributed server model for implementing additional monitoring on select individuals as guided by the organization's monitoring policy. Individuals who are considered a higher risk or are suspected of malicious insider activities may have their internet browser on their workstation computer configured to use a specific proxy server designed to detect and prevent data exfiltration, such as the methods described in this technical report and *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012]

Before implementing any type of monitoring program, organizations must consult with legal counsel to develop a program that is lawful.[5] The organization must disclose to all employees and trusted business partners that all activity on employer-owned devices and systems is being monitored and that employees should have no expectation of privacy in this activity, subject to applicable laws.

The disclosure should also state that the contents of encrypted communication are also subject to monitoring. Login banners on all entry points into information systems, including workstations, network infrastructure devices, and VPN connections, should be implemented. These login banners should clearly state that all communication, including encrypted communication, are monitored and privacy is not afforded.[6]

5.2 Intellectual Property Repository Security

As described in this report, the Insider Threat Center created an IP repository on the ICAP server. This approach requires an administrator to maintain the repository of IP on each server. This approach can create administrative overhead and present risks to the organization. The servers that contain sensitive IP must be properly secured to prevent unauthorized disclosure because these repositories represent a central storage location that could be the subject of attack by malicious insiders or even external parties.

[5] Any suggestions in this report are based on the U.S. Privacy Framework, available online at http://www.whitehouse.gov/sites/default/files/privacy-final.pdf

[6] The advice in this section about consulting legal counsel, disclosing monitoring activities to employees, and establishing login banners was originally published in the technical note *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012].

Organizations may wish to consider allowing the GreasySpoon ICAP server the ability to index file shares on other servers to reduce the risk of having IP stored in multiple locations with different security levels. In a Microsoft Windows environment, this strategy can be achieved by using Samba or the Common Internet File System (CIFS) to share files between a Microsoft Windows and Linux system.[7] The indexing service should be able to connect to the file servers with a user account that has the privileges necessary to access repositories. This approach allows the organization's information security and information technology teams the ability to leverage a centralized permissions system (e.g., Active Directory) to control access to the organization's sensitive information.

5.3 GreasySpoon ICAP Server Security

The ICAP server processes and stores sensitive information. Because this implementation of the GreasySpoon ICAP server works with the methods described in *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection*, end users will believe that their communication is always encrypted [Silowash 2012]. However, data passing through the proxy and onto an ICAP server can be viewed as plain text because the proxy breaks the SSL web session and re-establishes it to aide in inspection. Therefore, the proxy and ICAP servers must be properly secured to prevent unauthorized access.[8]

It is technically possible to use this ICAP server to intercept and record secure transactions, so additional physical and logical mechanisms to prevent misuse are required.

The organization should consider following accepted Linux best practices for securing the ICAP server. The following are additional considerations to further secure the system:

1. Provide any administrator who has access to the proxy server with additional training that identifies what he or she can and cannot do with the system. The training should address privacy issues as well as issues identified by the organization's legal counsel, which may include issues related to regulations, laws, and company policy.
2. Allow only a limited number of users that have a valid business need and need to know to access the proxy server.
3. Permit remote administration only from a management network using encrypted protocols, such as SSH with two-factor authentication. SSH can be configured to use password and certificate-based authentication.
4. Configure a host-based firewall to permit connections only to essential services, such as the ICAP server (TCP 11344 in this technical note), SSH, DNS, HTTP, and HTTPS. The ICAP server firewall should be configured to only permit TCP 11344 communications from other proxy servers.

[7] Further information about using Samba or CIFS to mount Microsoft Windows server shares within Ubuntu is available online at https://help.ubuntu.com/community/MountWindowsSharesPermanently.

[8] The advice in this section is adapted from the previously released technical note *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection* [Silowash 2012].

6 Use Case

To demonstrate the effectiveness of this control, we use an existing case example from the MERIT database to show how the insider's successful exfiltration attempt could have been blocked by the control described in this report.

6.1 Actual Case

Consider the following actual case:

> *The insider, a foreign national, was employed as a technical operations associate by the victim organization, a pharmaceutical company.*
>
> *During the course of his nearly three-and-a-half-year employment, the insider systematically and continuously downloaded the organization's confidential and proprietary information. The insider planned to use the information to start a competing organization in his home country. While on-site and during work hours, the insider downloaded several gigabytes of information, including 1,300 confidential and proprietary documents. The insider emailed some of these documents to potential foreign investors.*
>
> *The incident was discovered when an internal audit of the insider's computer revealed the insider's application to start a beneficiary organization in a foreign country. During the investigation, a conversation was taped between the insider and a potential foreign investor. This conversation led investigators to a meeting between the insider and a potential foreign investor. During this meeting, the insider showed the investor some of the organization's trade secrets, including those related to a drug that the organization spent more than $500 million to develop. The insider was arrested and convicted.*

By implementing the methods described in this report, the organization could have likely been able to detect and block the data exfiltration much earlier. The following sections describe the events that might have occurred if this control were in place at the time of the incident.

6.2 Intellectual Property Creation

During the organization's business activities, IP is created in the form of PDFs and Microsoft Word documents and saved into the IP folder being monitored by WebDLPIndexer. WebDLPIndexer immediately parses the text out of these documents, stores and organizes the text in the Apache Lucene index, and creates something called a document vector, which is a way of mathematically representing the document's content so that it can be compared against other documents' contents.

6.3 Data Exfiltration Attempt

The insider downloads 1,300 confidential documents from the fileserver, each of which has already been indexed by WebDLPIndexer. The insider selects a few of these documents to email to foreign investors, adding them as attachments to a message through his personal Gmail account. However, the attachments are failing to upload. Unbeknownst to the insider, the organization has implemented the control described in the technical note *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection*; the control has detected his exfiltration attempt, proactively blocked it, and logged the event [Silowash 2012]. Before the organization's security team has a chance to organize and respond to the insider's actions, he decides to go the simpler route and paste the various sections of the documents' contents into the body of the Gmail message. He clicks *Send* and waits for the message to leave the organization.

6.4 Detection and Response

Because the insider's web browser is configured by group policy to route all traffic through the internal Squid proxy, the outgoing web request arrives at the proxy and is eventually redirected to the GreasySpoon ICAP server. The ICAP server receives the message and runs the WebDLP script shown in the Appendix.

The script forwards the text in the body of the outgoing web request to WebDLPIndexer, which happens to be running as a background process on the IP server. WebDLPIndexer receives the text, uses a cosine similarity algorithm to compare the request against each document in the index, and finds that a few of the documents in the index score a 50% match against the outgoing request.

This result is returned to the WebDLP script, which is programmed to identify that any requests that match above the organizationally defined threshold of 25% should be blocked. It erases the text in the body of the outgoing request, logs the event, and forwards the request to Gmail. The Gmail web application does not expect to see a blank request, and returns an error message to the insider's browser.

Meanwhile, the logged event is pulled into the organization's log management system or security information and event management (SIEM) system, and the administrator is further alerted to the insider's activity.

6.5 Debriefing

The technical defensive control exhibited in this technical note provides an effective way to detect and actively respond to the exfiltration of important textual IP over a web-based email service. As shown, such a control would have easily detected and prevented the insider in the example case from getting as far as he did in his scheme to steal valuable company assets.

Appendix

```
//--------------------------------------------------------
//==ServerScript==
//@name WebDLP
//@status on
//@description Intercept requests on some sites and return a 403 forbidden response
//@include .*
//==/ServerScript==
//---------------

import java.io.*;
import java.util.*;
import webdlpclient.*;

private static final String logFileName = "log/WebDLPClient.log";
private static final int threshold = 20;

public void main(HttpMessage httpMessage) {

    if(httpMessage.getUnderlyingBytes() == null) return;

    byte [] txtBytes = httpMessage.getUnderlyingBytes();
    String txt = new String(txtBytes);
    txt = txt.replaceAll("%20"," ");    //Example of sanitization.  Especially effective for Gmail, for example

    if(!txt.equals(null) && !txt.equals("")){
        Logger.logFile = new File(logFileName);
        try {
                WebDLPClient client = new WebDLPClient(logFileName);
                HashMap<String,Double> map = client.testSimilarityWithThreshold(txt, threshold);

                if(map.size() > 0){
                    Logger.warn("User request has been blocked for exceeding the threshold of "+threshold+"!");
                    Logger.log(httpMessage.getRequestHeaders() + "\n" + txt);
                    httpMessage.setBody(" ");
                }
            } catch (IOException e) {
                Logger.exception(e);
            }
       }
}
```

References

URLs are valid as of the publication date of this document.

[Apache 2012]
Apache. *Apache Lucene Core.* http://lucene.apache.org/core/ (2012).

[Silowash 2012]
Silowash, G.; Lewellen, T.; Burns, J.; & Costa, D. *Detecting and Preventing Data Exfiltration through Encrypted Web Sessions via Traffic Inspection* (CMU/SEI-2012-TN-011). Software Engineering Institute, Carnegie Mellon University, 2012. http://www.sei.cmu.edu/library/abstracts/reports/12tn011.cfm

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY**
(Leave Blank)

2. **REPORT DATE**
October 2013

3. **REPORT TYPE AND DATES COVERED**
Final

4. **TITLE AND SUBTITLE**
Insider Threat Control: Using Plagiarism Detection Algorithms to Prevent Data Exfiltration in Near Real Time

5. **FUNDING NUMBERS**
FA8721-05-C-0003

6. **AUTHOR(S)**
Todd Lewellen, George J. Silowash, and Daniel Costa

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. **PERFORMING ORGANIZATION REPORT NUMBER**
CMU/SEI-2013-TN-008

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
n/a

11. **SUPPLEMENTARY NOTES**

12A **DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified/Unlimited, DTIC, NTIS

12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**

In organizations with access to the internet, the potential for data leakage is ever present. Data loss prevention is a difficult issue because exfiltration channels, such as modern webmail services, are readily available to insiders. An insider can paste text into a webmail message to bypass other controls. Therefore, monitoring must include the content of this communication. A data loss prevention control determines if the content in outgoing web requests is similar to the organization's intellectual property, actively blocks suspicious requests, and logs these events. This technical note describes how a control can monitor web request traffic for text-based data exfiltration attempts and block them in real time. Using this control can help an organization protect text-based intellectual property, including source code repositories.

14. **SUBJECT TERMS**
data loss prevention, data leakage, data exfiltration, controls, intellectual property

15. **NUMBER OF PAGES**
34

16. **PRICE CODE**

17. **SECURITY CLASSIFICATION OF REPORT**
Unclassified

18. **SECURITY CLASSIFICATION OF THIS PAGE**
Unclassified

19. **SECURITY CLASSIFICATION OF ABSTRACT**
Unclassified

20. **LIMITATION OF ABSTRACT**
UL