DeCore: Detecting Content Repurposing Attacks on Clients' Systems

Smitha Sundareswaran, Anna C Squicciarini
College of Information Sciences and Technology
The Pennsylvania State University
{sus263,acs20}@psu.edu

Abstract. Web 2.0 platforms are ubiquitously used to share content and personal information, which makes them an inviting and vulnerable target of hackers and phishers alike. In this paper, we discuss an emerging class of attacks, namely content repurposing attacks, which specifically targets sites that host user uploaded content on Web 2.0 sites. This latent threat is poorly addressed, if at all, by current protection systems, both at the remote sites and at the client ends. We design and develop an approach that protects from content repurposing attacks at the client end. As we show through a detailed evaluation, our solution promptly detects and stops various types of attacks and adds no overhead to the user's local machine or browser where it resides. Further, our approach is light-weight and does not invasively monitor all the user interactions with the browser, providing an effective protection against these new and powerful attacks.

Keywords: Content Repurposing, Malware, Web 2.0, Same Origin Policy, Information Flow.

1 Introduction

The emergence of Web 2.0 has brought with it an upsurge in the use of Web applications and Web-based communities that allow their users to load, store and share their content with others. These social computing platforms are an easy target of hackers and phishers alike, to whom the user content represents a wealth of information.

User uploaded content may potentially include executable files or malware, which have then the ability to access any other content which resides in the site's domain. Malicious files may harvest users' remotely stored sensitive data, and send them back to the hackers who triggered the attack. Further, when such malware is opened on the browsers of the users, it has the ability to access all the information present on their local machines, such as cookies or password files. To prevent these attacks, Web-sites often prevent users from uploading any executables, such as EXE files, or files which may potentially contain executables, such as XML files. These restrictions can be overcome by subverting the legitimately allowed uploadable file types such as images and text files to contain within them other executables. These attacks are referred to as *repurposing attacks*, and are nowadays proliferating. In fact, a number of attack vectors can be crafted to exploit this vulnerability such as botnets [20], different forms of distributed denial of service attacks [28, 2] and various forms of malware exploring the internal structure of the Web 2.0 platform.

Content repurposing has, however, often been used to allow a particular type of file to carry more information than it otherwise would. Examples are steganography [30], where the image is modified to carry messages, and mimic functions [29], where a file is modified to have the statistical properties of another file type. Hence, content repurposing has very important and legitimate uses, even with respect to security of files. A trivial solution such as never allowing repurposed content to be opened can be detrimental to the usability of these methods.

In this work, we thoroughly investigate the effect of misusing content repurposing. We conduct a preliminary analysis focusing on two specific examples of repurposing attacks, which have gained attention from the media [4, 3]. The first attack vector, namely the Gifar [4], uses a form of steganography, combining images or any other file types (such as word file or flash etc) with Jar files. The modified file is used to carry the payloads of various attacks that can be triggered when posted on Web portals [9]. The second attack vector consists of repurposing a Flash-file by modifying its ActionScript and combining it with other file types enabling it to be uploaded to any online content management sites [3]. While some patches have been proposed for the Gifar attack [25], users need to update their system by installing the latest version of Java to install the patch. This is often cumbersome for end users [18] and hence may not be a suitable solution. Further, while specific solution exist for certain attacks, we are unaware of any general solution addressing the class of content repurposing attacks. For example, to date Flash-based attacks have not been patched.

We present a slew of attacks which can be launched easily using Gifars and Flash-file. These attacks help us demonstrate the ability of the repurposed content to manipulate and steal information from the local machine of the victim, when these files are opened in the victim's browser. Existing defense mechanisms do not recognize the repurposed files as malicious, nor do they raise an alarm when the attacks are carried out. Our tests also demonstrate that the Gifars and malicious Flash-files can be uploaded to numerous popular Web-sites, including Picasa, Orkut and Friendster. Surprisingly, even common antivirus or antispyware fail in detecting an ongoing attack.

In light of these observations, we propose a new approach to protect against generic content repurposing attacks. Our approach is to silently monitor the content being served to determine if it is repurposed, and subsequently determine whether the events occurring signal an ongoing attack. An attack is detected based on the analysis of the control flow graph, which given a set of inputs and the current state, can be used to predict possible legitimate actions and thus identify illegitimate states. To capture the users' interactions with the browser we rely on DOM (Document Object Model) Events [8], since the DOM forms a representation of the Web page as shown to the user and accepts asynchronous input from the user.

We design the **DeCore** (**De**tecting **Co**ntent **Re**purposing) system using a client-end architecture, since it effectively allows us to monitor the user's interactions with the browser without invasively monitoring the specifics of the input. Further, if the protection is at the server-end, the attacker can overcome server-based protection by hosting the malicious file on a remote server and launching an attack on the end user's system by tricking the user into clicking the link which launches the applet in the malicious file.

We deploy the DeCore using an add-on for Mozilla Firefox, and Google Chrome. As demonstrated by our test results, the add-on adds no overhead to the users local machine or browser where it resides. It also does not invasively monitor all the user interactions with the browser, in that it is not concerned about specific clicks or other input by the user such as text, user ids or passwords.

The rest of the paper is organized as follows. In the next section, we elaborate on the content repurposing attack, and discuss its applicability in existing Web-sites. In Section 3 we present the design of the DeCore followed by the system's implementation in Section 4. DeCore's evaluation and testing are discussed in Section 5. We discuss related works in Section 6 and conclude in Section 7.

2 Content Repurposing Attacks

In this section, we describe how content repurposing attacks can harm users' systems and remote servers, by focusing on two representative types of attack vectors and on a few examples of attacks. Next, we discuss how the current protection mechanisms fare against these attacks.

2.1 Overview of the Content Repurposing Attacks

Content Repurposing attacks take some particular type of content or file type and then modify it by combining it with active file types which contain executables. This maliciously crafted content remains undetected for two main reasons. First, the repurposed content masquerades itself as a benign file. Second, the operations performed by a repurposed file when it is loaded in the browsers are often the same type of operations needed by the Web applications to genuinely perform their tasks. Two popular types of content repurposing attacks are Gifar-based attacks [4] and Flash-based attacks [3]. While our analysis is intended to be general to all content repurposing attacks, we conduct our preliminary investigation with these attacks in mind, since these are the most recent and harmful attacks identified. Other important attacks falling under the umbrella of content repurposing are the recently announced attack utilizing zip files along with steganography to launch malware via emails [22], and attacks on Flash crossdomain policy files, and sniffing the MIME with images in Internet Explorer [25].

Both Java applets and Flash-files leverage the same origin policy (SOP) in Web browsers. The SOP governs access control among different Web objects (such as HTTP cookies, HTML documents, images, JavaScript, CSS files, XML files, etc) and prohibits a Web object from one origin from accessing Web objects from a different origin[17]. By exploiting this rule, the attacker can upload content able to access all data and files on the domain the repurposed content is served from. The malicious content can even be given the capability to browse through the internal network structure of the domain it is uploaded to and also to attack the local machine of a user via the browser.

In their most common form, Gifar-based attacks exploit the fact that when an image file, such as a *.jpg or a *.gif file, is combined with a JAR file, the resulting file can be rendered as a valid image by the browser, while the Java Virtual Machine is capable

of recognizing the same as a JAR file. The JAR files contained in the Gifars are applets, which can be used to exploit the victim whose browser the Gifars are running on. Specifically, the Gifar is created when the attacker combines some malicious applet in the form of a JAR with an image using the command line's copy command. For the attack to be completed, the attacker needs to be able to invoke the applet using an HTML file, like the one shown below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body bgcolor="#dddddd">
<applet code="localfile.class"
archive="file:///C:/Program%20Files/PostgreSQL/EnterpriseDB%20ApachePhp
/apache/www/ drupal/sites/default/files/images/gifar2.gif"
width="100" height="100">
</applet>
</body>
</html>
```

The browser then opens the image containing the applet as a JAR and executes the code in it. The HTML file includes the Gifar the same way any non-malicious applet is usually included, except that the applet tag refers to the Gifar file.

Gifar-based attacks succeed for a number of reasons. First, the Java Runtime Environment does not check the extension of files before parsing the JARs. Further, browsers run any file in the format specified by the underlying HTML code of a given Web page without verifying what the actual extension of the given file is. Third, the other underlying vulnerability which allows all these attacks to succeed is the fact that the most Web portals allow unverified traffic to flow through it.

Flash-based attacks are similar in that they also exploit the fact that the type of file rendered is not verified by the Flash plugin, and the ActionScript used by Flash-file can be used to execute malicious code. The attackers therefore combine malicious Flash-files with any type of zip files or even poorly formed image files, self-extracting executables, Microsoft Office Open XML documents, XPI files, and, even JAR files. The files are combined similarly as in the case of Gifar-based attacks, that is, by using the command lines copy command.These files can then be uploaded to a large number of sites, while remaining undetected. Zip files, for example, can be sent as attachments or uploaded onto any Web-site which stores such files. In order to be executed, these files are simply passed to an Adobe Flash Player, or in case they are sent as attachments, they simply need to be opened by the recipient.

The attacks carried out by content repurposing attacks target integrity, confidentiality and availability of the user's data. We now provide a few examples of attacks, grouped by the security property being violated.

Attacks to confidentiality: Attacks breaching confidentiality usually circumvent security mechanisms which protect user's data. To accomplish this task, content repurposing attacks can be launched to bypass the control of certain authentication protected information in various ways. First, the Gifar can be used to bypass Web-sites' authentication. The user will have to download the JAR file which completes the attack, that can be delivered using a Gifar image. This Gifar retrieves the saved cookies of the user and subsequently uses them for login, using a second program. The program sending the cookies to the attacker is the one which is delivered to the victim in the form of the Gifar. The victim also needs to "launch" this JAR by clicking on a link. This could be the link to an HTML file on a remote server, which has been posted on a social network (SN) site. In order for the cookies based authentication to succeed, the victim should have persistent cookies (i.e. the "Remember me" option is selected on the browser for any sites which require a password-based login). This attack can also be adapted to send all the saved passwords of the victim to the attacker. If the cookies are persistent, then the passwords are stored in easily reachable files in the end-user's local machine. For example, in Mozilla, the saved passwords Firefox are stored in a file called "signons3.txt" (this file varies by the version of the browser the victim uses) -and in `Microsoft Credential File` in Internet Explorer-. This file combined with the "key.db" file of Firefox can be forwarded from the victims system to the attacker in order to allow him to gain all passwords. The same attack can be perpetrated with Flash-files, where the attacker needs also to ensure that user logs in to their account after the malicious file has been loaded onto their browser.

Another type of attack which poses a threat to confidentiality is a remote intrusion attack. By using a repurposed file, the attacker can open an explorer window which allows him to explore the victim's machine from his remote location. In case of a Gifar, the JAR that allows him to open the window exploits the Java Remote Desktop (JRD) so as to provide the attacker a control of the window. The attack begins when the victim opens the Web page which embeds the Gifar image as an applet. The applet then executes and runs the JRD using the Runtime.exec() function, which opens a remote window and connects to the attacker's, allowing him to remotely explore the victim's system. In case of a Flash-based attack, the ActionScript file can be used to exploit the JRD in a similar fashion. In this case too, the attack begins when the Flash-file is loaded onto the browser.

Attacks to availability: In the context of content repurposing, attacks to availability are of two types. One of them is command and control (C&C), where the attacker tries to take over the victim's system by using it as a bot. A C&C attack basically allows the attacker a surreptitious channel to install and execute the files which turn a machine into a bot. The C&C channels are used by the attacker to remotely control the botnets [10]. The Gifars or Flash-files could provide the attacker a distributed C&C channel for the botnets owned by the attacker. The attackers can easily create and embed their server and/or client programs as the JARS or ActionScript files, such that once the HTML page invoking the applet embedded in the Gifar is loaded or the Flash-file is opened in the browser, botnets receive their commands and begin to carry out the malicious operations. The other type of attack is a form of denial of service, where the attacker tries to choke the victim's browser. The JAR file included in the Gifar launches a series

of windows to the victim's profile. The page being opened can be a page on the Web site that the attacker wishes. In case of Flash-based attacks, similar actions are carried out by using an ActionScript based code. This attack can further be modified into a DDoS attack. The Web server hosting the Gifar or the malicious Flash-file can be subjected to a DDoS if the attacker posts a number of Gifars (or Flash-files) on different profiles and also sets the number of windows being opened sufficiently high. The attack can be made more disruptive by choosing a page with "heavy" elements like multiple multimedia files.

Attacks to integrity: These attacks aim at changing the content of some of the user's files, and may result in the victim's corrupted data, or in a denial of service of sorts. For example, if the attacker modifies the password files of the victim, when the victim tries to log in using a saved password, the authentication would fail. The attacker can additionally issue the commands in the C&C attacks by modifying files stored on the victim's local machine. Another attack which falls under this category is when the attacker tries to modify the remote profile (say in a social network site) or web space of a user. For this attack to succeed, the attacker first needs to bypass the authentication, which constitutes another attack in itself as discussed below.

To assess the potential of content repurposing attacks, we have extensively tested the attacks in real-world settings. We successfully tried Gifar-based attacks on Orkut, Friendster, LiveJournal, Facebook, the art community DeviantArt [1]. These sites allow us to load the Gifar, directly or indirectly via remote links, and the Gifars are stored without modification.

To test Flash-based attacks, we combined Flash-file with image and zip files. DeviantArt allows both uploading modified images and embedding the files directly, as does Orkut. Both DeviantArt and Orkut are the most susceptible to these types of attacks. As with Gifars, Facebook was one of the most resilient sites against the attacks. However, the malicious Flash-files can be directly embedded on a page, such as profile page, or even by including them as part of an HTML based post by using "fb: swf" tag. Therefore, it is not fully immune against such attacks. In LiveJournal, we cannot embed the Flash-file directly, however, we can upload a modified image. Though the attack is not launched without a Flash player (thus making LiveJournal the safest), it leaves a vulnerability waiting to be exploited. Further, Flash-based attacks have been successfully conducted also against email systems, such as Gmail [3].

Finally, we tested the top 5 Antiviruses and the top 5 AntiSpyware [2] as listed by CNet [6,5], and found that none of these softwares detected any of the Gifar files as malicious, nor were they able to recognize the attacks when they were actually going on. The Antiviruses fail to recognize these attacks because these attacks perform functions which are usually carried out by the browser while loading certain pages. For example, the file modification based attacks are not easily recognized because the `Password`

[1] LiveJournal is available at www.livejournal.com, Orkut at www.orkut.com. Facebook's site: www.facebook.com, DevianArt is available at www.devianart.com

[2] The programs tested by us were Lavasoft -Ad Aware, Zone Alarms, Tenebril Spycatcher, Webroot Spy Sweeper (SpyCtacher Express -5.1.2), SpywareDoctor, Symantec Endpoint Protection, Kaspersky, Norton Antivirus, BitDefender, F-Secure Antivirus and Avast.

`Files` are modified whenever the user changes a password or asks the browser to remember another password. The AntiSpywares do not recognize content repurposing attacks because the attacks do not necessarily require any visit to malicious sites or to carry out other suspicious activity like displaying advertisements or scanning for personal user information.

2.2 Existing protection mechanisms

Current systems try to cope with content-repurposing attacks in various ways, both at the server end and the client end. However, none of these approaches is satisfactory, as they all suffer from some significant vulnerabilities. Below is an overview of the most common attack defenses currently implemented.

- Using a "throwaway" server for images. That is, the images and other user-uploaded content are stored on a separate server which is not on the same domain as the rest of the content. This approach thwarts content repurposing attack which exploit the SOP as it does not allow for the SOP to be valid. However, this solution can be adopted only by large popular portals as it is not cost effective for smaller ones. Besides,the malicious repurposed files can still be uploaded to some remote site which is owned by the attacker and the attack can be launched the end user's local machine from that site.
- Ensuring that only authenticated scripts can run in the server space. The server can require any script which runs on the server side or searches the database to be authenticated to it. This however does not ensure that authenticated scripts do no leak data. It also does not prevent the repurposed malicious files from using the stored cookies of the client or running scripts on the client's system when the page hosting the malicious content is visited.
- Scrubbing the images when uploaded. Filtering any content which is being uploaded to a Web application end involves eliminating any associated data with the images such as any metadata and also stripping the images of any code embedded in them. Such filtering of content can be performed at the client end when the content is being uploaded, as is done by Orkut, or it could be done after the content is uploaded to the server. This technique applies only to Gifar-based attacks; its very difficult to remove the content attached to zip files, since zip files as such are meant to carry other file types. Resizing images often causes the embedded code in the Gifar to be corrupted or lost. While filtering the content before it is saved ensures that no malicious content is saved on the server, this approach could also result in certain types of animation or multimedia files being corrupted or spoiled otherwise as these files often have some sort of associated code in Java, JavaScript or PHP. Besides, scrubbing may not always be sufficient to completely remove all the malicious content attached with the image; a sophisticated attacker would be able to still launch the attack by restoring the corrupted content.

Additionally, there are some easy-to-implement 'shortcuts' solutions [14, 25], such as avoid the use of persistent cookies to prevent an attacker from bypassing Web-sites authentication. These approaches, however, cannot be deemed as practical solutions

because of the popularity of such persistent cookies. Web portals could also opt for limiting/blocking the HTML links posted on their sites. This, however, is not a suitable solution, since the ability to post arbitrary comments contributes largely to the popularity of many content management portals.

As we return later in the paper, using secure browsers like Chrome does not hinder content repurposing attacks because these browsers do not verify whether the content being served is legitimate with respect to the plugins of the applications they are being served by. Verifying the integrity of content uploaded at the front end or ensuring that applications can only access data legitimately required by them is not sufficient either. These checks can be easily circumvented by attackers who can always use different applications to upload malicious content and further attack applications to leak any data legitimately gathered by them.

At all effects, what we are trying to tackle is an information flow problem rather than simply an information integrity one. Hence, our approach is to detect the information flow violations between the targeted Web site, the local systems and any external Web site which is loaded while the original site is being viewed.

3 The DeCore System Design

To protect from repurposing attacks, we have designed a protection mechanism, referred to as *DeCore* (Detecting Content-Repurposing). While our protection system implementation is primarily tailored for the known attacks described in Section 2.1, the DeCore system design is modular, and constitutes a general protection mechanism for both the victim's local system and, to a certain extent, his remote data. The overall design principle of the DeCore system is to monitor the host's observable properties, such as internal state, state transitions (events), and I/O activity to detect and zero-in possible attacks. The DeCore can be successfully deployed at the user-end, or as a component at remote server. Since most of the content repurposing attacks aim at attacking resources on a end-user's machine, however, a client-based solution offers a higher degree of visibility as it is integrated into the host it is monitoring, as an application.

Our architecture is characterized by two main logical components: the *auditor*, and the *detector*.

The DeCore flow auditor The DeCore System's Auditor is responsible for sensing an ongoing attack. To this extent, its main task is to detect anomalies in the information flow rules that are originally intended by the Web Portal which is being accessed by the user. These anomalies can either be with respect to the content being served or the expected flow of operations.

The auditor detects anomalies by carrying out three main operations: (i) verifying whether all the files being served on a page have the legitimate extensions supported by the plugins, i.e. a Java Plugin is served only a JAR file while a Flash Plugin is served only files with extensions *.swf, *. fxg, *.fla etc. (ii) capturing all the interactions between the user and the browser (iii) matching these interactions with the changes in the files at the end-user's local machine and also checking the displayed content at the Web server's site.

Task (i) is completed by referring to a list of legitimate plugins, and then checking whether the file type being served is the same as the one requested for. To obtain a list of legitimate extensions supported by each of the plugins, the monitor periodically searches the Web for a list of all possible extensions.

To carry out tasks (ii) and (iii), the auditor relies on a *control-flow graph* (CFG, for short). The CFG is a finite labeled graph, constructed upon the user opening a given Web-site. The CFG captures all the possible interactions between the browsed Web site, the end-user's machine and a remote site in order to identify flows which result in potentially malicious code being run. The nodes represent various possible states the browser can be in and the edges represent the required user input to make a transition between two legitimate states. The CFG is derived by considering all the possible DOM events and JavaScript links by the DeCore after examining the source code of a Web-site.

Example 1. Figure 1 provides an example of a control-flow graph between a blog, the user's local system and an external Web site's domain. In the figure, the edges which are not crossed denote legitimate flows. A crossed edge between the comments or messages, and any entity indicates a malicious operation being performed.

For example, the edge F5 signals a new Web site being opened as a result of the user's click on the blog. A blog could contain a link to a legitimate Web site. Therefore when a connection is launched by clicking the link on the scrap to the external Web site, it may not necessarily be any malicious activity. However, when the external Web site interacts with the user's local machine files using any I/O operation (F6), it signals that an external entity is trying to modify something on the user's local machine.

Once the graph is loaded, the auditor calls the detector which verifies that the information flows from and to legitimate states as prescribed by the graph.

The DeCore Detector At the heart of the DeCore is the detector. This component interprets CFG violations and reported events from the auditor, and decides whether or not an attack is undergoing. If the system has been compromised, the detector is responsible for responding in an appropriate manner.

The detector has, in turn, several logical subcomponents, each of which checks whether a given type of attack is under progress, and takes some action to either prevent the attack or block it, and to alert the victim. Due to the polymorphic nature of repurposing content attacks, a single approach may not suit all the possible ways according to which this attack vector is exploited. Therefore, similar to an intrusion detection system, with DeCore it is possible to implement several security policies, zeroing-in the different forms of these attacks.

Each policy module leverages the CFG developed by the auditor and runs in tandem with it to detect a particular type of attack. Policy modules can be run stand-alone or in concert with other policy modules. We provide a discussion of three sample policy modules by classifying the attack according to the security property being violated. We chose these policies as examples that illustrate more general paradigms of policy design that can be supported by this architecture.

Attacks to the Integrity: The DeCore System's security policy towards attacks on integrity is to constantly poll the user's data and to notify the user of any seemingly

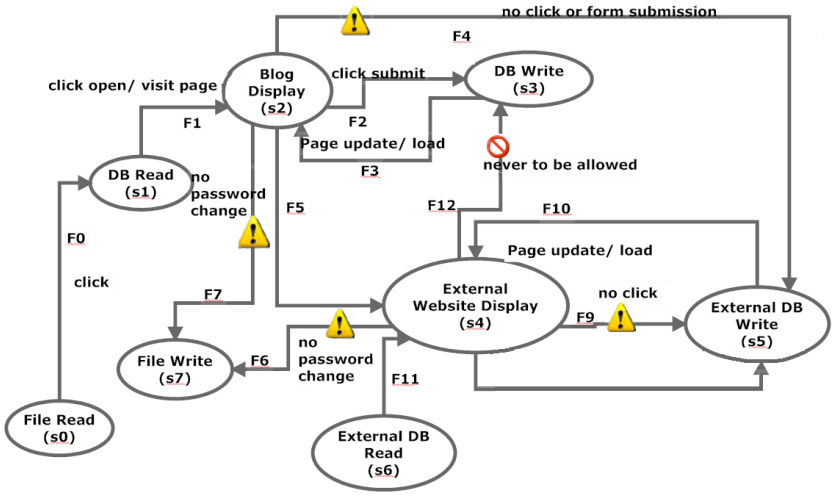

Fig. 1. Example of Control-Flow graph

illegitimate change to the data. An illegitimate change is differentiated from a legitimate one by checking if changes to the data take place without explicit input from the user. To avoid false positives in cases where such data may be updated without user interaction, the polling of the data is not done unless the data is located on some location which can be updated only by the user such as the user's local machine or a profile page in a SN or a closed blog.

Attacks to availability: The security policy applied in the case of attacks to availability are based on an event-triggered approach, where any event which can potentially disable the user's control on the system triggers an alarm which stops the event in question from proceeding without the user's approval. Events monitored by this module include the browser choking denial of service attack discussed in Section 2.1.

Attacks to Confidentiality: The security policy for this type of attacks involves monitoring whether any access to the user's data takes place once a page serving suspicious content is opened. Should such an access be detected, the user is notified, to indicate that an attack may be undermining the confidentiality of his documents.

4 The DeCore Implementation

To better understand the implementation difficulties, performance overhead, and practical effectiveness of our architecture, we implemented the DeCore System as a browser plug-in. We purposely encoded most of the add-on in JavaScript and using JAR files, so as to ensure its portability to any browser. All references to the files and file paths were left platform independent, thus making it compatible with different platforms and file systems. To port the DeCore onto a specific browser or OS, the files references must be configured according to the chosen platform. Further, the DeCore is well compatible

Algorithm 1 Algorithm to detect attacks launched using repurposed content

```
1: Send HTTP GET request
2: Response.type:= text/html
3: Enumeration headerNames:= request.getHeaderNames()
4: plugins − type: = pluginspage[type]
5: array extensions [] = "http://www.google.com/search?q=".plugins-type."+extensions"
6: for i ← 1 to length(extensions[]) do
7:    if extensions[i] == headerNames then
8:       status:= "No Attack"
9:    else
10:      status:= "Attack Suspected"
11:   end if
12: end for
```

with sandboxed environments, such as Google Chrome. As long as the source code of the Web-site is visible to the add-on, the DeCore can monitor the response obtained by the HTTP Servlet which allows the getHeader method to obtain a valid response. Therefore it can detect when any repurposed content is being obtained in response to the request.

4.1 The DeCore Auditor Implementation

The DeCore Auditor checks whether the file being served is repurposed by verifying that the file has an extension type supported by the plugin requesting it.To obtain a list of legitimate extensions supported by each of the plugins, the monitor periodically searches the Web for a list of all possible extensions. It determines the type of file being served to the page by running a small Java-based program which sends a request to the domain serving the files on behalf of the Web site using the getHeader method of the HttpServletRequest. The pseudo code for this type of validation is presented in Algorithm 1.

As content repurposing attacks are carried out by completing a few seemingly normal events such as redirection to an external Web site from a source site, or reading of the password file, the auditor verifies that none of the possible states modeled by the CFG is reached without the DOM events which are needed to ensure a legitimate transition to the given state. The DOM is a platform-independent, event-driven interface which accepts input from the user and allows programs and scripts to access and update the content of the page. The CFG itself is derived by the DeCore using a Java program which reads each line of the source code of a Web-site, considers all the JavaScript links, buttons, boxes and form elements, and HTML links, buttons, checkboxes and form elements to derive the CFG. The CFG takes into account all the possible actions which require a user's input and any actions which result in redirection to another page

or the opening of a new window or tab. DOM events that are not caused by the user's interaction or input are indicative of a possible attack[3].

For example, the flow monitor checks that all the page load and window load events are actually caused by other DOM events such as mouse clicks. The mouse clicks indicate a user's interaction with the elements on the Web browser.

If any of these two conditions are violated, that is, if the states in the CFG are reached without the required DOM events or there is a violation in the flow, then an attack is assumed to be ongoing. Further, to improve detection accuracy, the auditor, besides correlating DOM events, checks whether DOM events such as keystrokes and mouse clicks are carried on at a legitimate rate for a human user. While it is possible for an attacker to simulate such keystrokes at a reasonable rate, these attacks would entail an unlikely level of sophistication. Such simulation requires the use of sophisticated HCI models such as GOMS and UIMs besides a huge database of similar activity by a human being [15, 26]. Finally, in order to minimize possible attempts of this type, the system requests feedback from the end user upon detecting an attack. For example, DeCore notifies the user when a Gifar attack is suspected through event-delivery notifications. These methods are discussed in detail below.

4.2 The DeCore Detector Implementation

We have four separate JavaScript components which enforce the security policy modules discussed in Section 3. Two of three sample policy modules of the detector consist of an individual JavaScript component (i.e. a single file) that leverages the detector framework, while one policy module is implemented using two JavaScript components. The implementation details of the security policy for each module is given below.

Attacks to the Integrity: Our integrity checker attempts to detect if the victim system files are being modified by periodically using `nsIFile` functions [23]. A `nsIFile` instance allows for a cross-platform representation of a location in the file system. Once an nsIFile instance is created, it can be used to navigate to ancestors or descendants of a given file or directory without having to deal with the different path separators used on different platforms. It can also be used to query the state of any file or directory at the position represented by the `nsIFile` and create, move or copy items in the file system independent of the platform on which the file is located. An nsIFile can be retrieved by either instantiating an nsILocalFile using a platform specific path string or by using cross-platform locations retrieved from the directory service. This approach is particularly well suited to securing files across multiple OS without intrusive monitoring of the user's file system. For example, if files are downloaded from the site which is suspected to host a Gifar attack, and not correlated to the user's event, the attack is deemed as started. The user is then alerted of a possible attack and asked to close the Web site hosting the Gifar. Further, if there is a change to the file system while the attack page is opened, the user is alerted to the changes.

[3] An exception to this rule occurs when the user has set some preferences which allow the browser to carry out some actions automatically, such as to automatically launch a prefixed number of tabs upon being opened the first time.

Another example of attack to the victim's integrity is as follows. The attack can target remotely stored user-generated content in social computing platforms, such as SNs, and blogs. Once the malicious file is opened, it can, for example, add spamming content malicious links or modify the user posted content. To prevent this attack, at the time the monitor suspects an attack, the detector periodically checks for unexpected (and not-user driven) modifications in the rendered content, while the Web page hosting the suspected Gifar is open. To limit the scope of the monitoring, the DeCore detects whether the page being served is the user's profile in case of a SN, or some closed site which cannot be modified without the user's input (such as his blog). Specifically, when one of such pages is accessed, a JavaScript-based component checks whether the last modifications occurred upon the user submitting a form, on the Web site, and matches the same to changes in the content. If the modification on the rendered content is not corresponding to some user input and a Gifar is being detected by the monitor, the user is alerted of a possible attack and asked to close the Web site hosting the suspicious Gifar. Notice that this approach in turn helps tracking whether the SN's database is being modified by some external code, while not requiring any interaction with the server, since the detection is based on data collected from the DeCore at the client-end. For other types of sites, the user can create a list of such sites that the DeCore should control, along with a list of sites to be excluded from the controls. In fact, the user can also indicate sites which by nature refresh dynamically their content (without generating DOM events), such as scoreboards or games, and therefore should not be monitored.

Attacks to availability: The module addressing the security policy for this attack takes a event-driven approach, by checking for page load events and mouse click events, and taking action upon certain conditions are verified. To avoid false positives, this detector's module checks whether the number of mouse click events are not only the same as the number of page events, but also that they were executed at the same rate as the loaded pages. If the difference between the mouse click and page load events, say x, is larger than a choking threshold μ (where $\mu << x$), an attack is deemed under way. A choking attack is also assumed to be going on if a very large number of pages (where the number of pages y is greater than a threshold β) is opened within a very short period of time (where the time is less than δ seconds)The user is then alerted to a possible attack and asked whether he wants to continue opening multiple pages.

Attacks to Confidentiality: The DeCore enforces the general security policy for such attacks by monitoring the file systems of the end user's local machine for any access. We use the `FileSystemWatcher` class in Java. This command is run in a loop till such time the window hosting the suspicious files is closed. The `FileSystemWatcher` class has an option called the `notifyfilter`. This option allows us to monitor whether the last access time of any of the files on a file system is changed. Should such a change occur, we notify the user, or depending on the user's confirmation, take more proactive actions such as encrypting the file. While possible to prevent access on the basis of the process accessing the files, we choose not to because doing this requires an invasive level of monitoring.

The JavaScript components used by DeCore are not dependent on the particular approach used by the attack but rather look for specific outcomes or effects produced by an attack. For example, for the choking attack, we check for multiple requests opening

multiple pages from the user's system. We do not check for specific pages being opened, nor do we check for the signature of a particular DoS attack. In this way, the DeCore covers the class of attacks where a victim's browser is rendered useless to him as it is taken over by a malicious script.

5 The DeCore evaluation

Our experiments were performed on a Dell Latitude D630 Laptop, with 2G Ram and a Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz processor. We conducted two separate set of tests. First, we tested the system's overhead. Second, we assessed the accuracy of the DeCore. Both the tests were conducted twice, once for Firefox and once for Google Chrome. The results presented below apply to both the add-on for both the browsers. This is because except the basic construction of the add-on itself, the rest of the code for detecting the various attacks does not change.

In the first set of tests, we compared the execution time for the browser with and without the DeCore add-on. We specifically recorded the time for opening new sessions with multiple tabs. We varied the number of tabs from 0 to 60 and the number of windows from 1 to 6. In order to ensure accurate results, each test run was carried out according to the following steps. First, we disabled the plugin, loaded a page over a quiescent network, and determined how long the page took to load. Next, we cleared the cache for the following run. When collecting data for the DeCore-enabled browser, the same methodology was used, but we first enabled the plugin, at each run. We reported no overhead caused by the DeCore and the exact same time was taken for the operations both with and without the add-on. The time required for Firefox to start was always around 1 ms. This time included only the time it takes for Firefox to start as a process by the system, and did not factor in the time taken to make the Firefox available for use. Further, we checked the maximum CPU usage and found that the difference in the percentage of CPU usage was less than $\pm$ 2 ms (for example, for 0 tabs with 1 window, when the Firefox session is being restored the usage was 47 % with the add-on and 46 % without the add-on. The usage for the Firefox session being restored with 6 windows and 60 tabs was 56 % without the add-on and 54 % with the add-on). We obtained similar results for Chrome. The major difference between the Firefox and Chrome testing was that in Chrome we cannot open multiple windows as in Firefox, so we just opened multiple tabs. Again, we reported no overhead in the case of Chrome either and the same time was taken for the operation with and without the add-on.

Our second set of tests aimed at verifying the accuracy of the DeCore. To this extent, we carried out several different experiments of increasing complexity on both the browsers. First, we begin with assessing false positive rates, i.e. whether the DeCore would falsely detect a page with benign Java and/or JavaScript components as a page hosting a Gifar. The tests were carried out by having the DeCore running while 100 different sites were visited to test the accuracy of our system when it is continuously monitoring for content-repurposing attacks. The sites were selected based on their popularity and on the presence of active components. The sites visited by us included popular gaming sites (such as *Games.com* and *Miniclip Games*), which often utilize JAR files and JVM to allow their users to run the games, magazines (such as *Elle* and *Glam-*

our) and blogs. With our second round of experiments, the page hosting the malicious files had benign components. Specifically, we created 100 sites, each of which embedded some variant of the attacks, such as the denial of service attacks or remote intrusion attacks. The actual attack code varied for each try, so as to create polymorphic attack code. To create the variations of the attack code, we introduced random NOP blocks in each attack to introduce random delays. Further, we combined one or more attacks with each other. Also, the page invoking the malicious content was different for each try. The elements we included in each page consisted of one or more of the following: images, videos, audio components such as wmv files, other benign JARs carried in applets but not embedded in images, text documents, hyperlinks, Java buttons, JavaScript buttons, JavaScript forms, zip files, Microsoft Office Open XML documents, XPI files, benign SWFs and simple games. The DeCore proved to be accurate in both set of tests, detecting the attacks correctly, regardless of the attack type. Finally, we created a new test case by launching multiple attacks at the same time. We crafted attacks so as to combine more than one content-repurposing attack on a single HTML page. We constructed the attacks in two alternative forms: we either hosted multiple repurposed files in a same page, or created a file which would carry out different attacks in a single file. Both Gifar and Flash-attacks were tested. With this attack, we were not only interested in checking whether the DeCore could identify and stop the launched attacks, but also whether the detection of one attack could slow down the detection of the subsequent ones. We tested 15 different attacks . The different types of attack were constructed by combining the attacks discussed in Section 2.1 with one another. We focused on some non-trivial attacks, namely five attacks with 2 repurposed files hosted at each page, five attacks with 3 repurposed files and 2 combinations of all the 4 repurposed files, resulting a total of 15 different types of attacks. We ran this experiment by hosting web pages on a secure remote PHP server and also on a server hosted on the same local machine where the DeCore system was deployed as a plugin. None of the attacks were successful. For example, the file modification attack was always detected with a delay less than 1 ms. Subtler attacks, such as bypassing the authentication, fail as well, since the victim's hard drives are always protected before the attack can be completed, thereby causing the attack to abort (hence, rendering the combination of attacks useless). The time required to complete any single attack to execute is (order of 100 ms) significantly higher than the time required for our detection script to run (order of 0.01 ms). The only delay was recorded when testing the choking attack in combination with 2 or 3 other attacks. Specifically, the attacks placing the choking attack as the last one being launched, resulted in the attack being started before any warning was raised by the DeCore. We found that unless the delay in detecting the attacks is a magnitude higher than 100ms (which never happens with out implementation), the chance of this attack being successful is negligible. Therefore, we conclude that the DeCore proves to be an effective protection mechanism, with respect to all types of content repurposing attacks.

6 Related Work

In this section we summarize some of the most closely related approaches recently proposed to thwart attacks similar to the ones we tackled in this paper. There are two

parallel lines of work that are of interest to us: monitoring-based systems [21, 12] and information flow control strategies [1, 13].

In [11] an approach similar to ours has been proposed for Ajax intrusion detection. The authors develop a monitor which matches if the series of requests received by the server is similar to an abstract request graph previously derived. While similar to our approach, Guha and colleagues focus on the response to the server from the client. Therefore, they mainly address server-based attacks, while the DeCore is geared toward attacks carried out at the client end. However, we also plan to enable our solution to detect server-based attacks in the future. Further, the proposed system needs to run as proxy between the server and the client, which evaluates the response from the client machine. Our solution is less invasive and does not rely on the response from the client to the server for its detection, thus succeeding at detecting attacks that affect only the client machine and provide feedback to the victim.

A similar approach to the above is taken in [7] by Dhawan et al. The authors develop a system which uses in-browser information tracking to analyze JavaScript extensions. We borrowed from this work the idea of using information flow by considering the DOM events, to investigate whether sensitive data is being leaked. However, Dhawan's approach is applicable to JavaScript Extensions, and it does not monitor the malicious behavior of any outside code, nor does it detect Java-based attacks. Also, unlike us, the implemented prototype requires modifications to the interpreter of Firefox, viz. SpiderMonkey.

The work by Karlof et al. also looks at attacks which sends the browser malicious JavaScript [17]. The authors focus on *Dynamic Pharming* attacks, that exploit DNS-rebinding vulnerabilities DNS and the name-based SOP to hijack a legitimate session after authentication has taken place. The solution presented, however, is completely different from ours. The authors propose two locked SOPs for web browsers. As opposed to the normal SOP, which regulates cross-object access control in browsers using domain names, the locked SOPs enforce access using servers' X.509 certificates and public keys.

Since content-repurposing attacks can be classified as stemming from information flow problems, the other way to tackle such attacks is by monitoring information flow. One of the widely accepted approaches to information flow monitoring involves using security typed languages such as JIF, Caml etc. JiF (Java - Information Flow) [13] is a security-typed programming language that extends Java with support for information flow control and access control, which is enforced at both compile and run time. Static information flow control could be used to protect the confidentiality and integrity of information as it is being manipulated by computing system. JiF can also be used to reduce the exposure of data to online organizations [13]. However in order for this approach to work, it is essential to know all the parties which are legitimately involved in an exchange and further to know what each party is allowed to receive. Since it is not possible for a third party application, which is situated at the client end, to know about all the information flow requirements without access to the SN's database or client input, this approach is not suitable.

A reference monitor, such as the Shamon architecture[21], has been often used to regulate the flow of information within the system and the between the processes. With

the use of remote attestation and virtual machines [12], the traditional guarantees offered by the reference monitor may be extended to provide the same guarantees on multiple machines, and thereby on the Internet scale. The disadvantage with reference monitors is that they are usually very heavy to implement due to their reliance on authentication. Further, they monitor all the system processes, but afford little help in maintaining the information flow in the browser. Sun released a patch to prevent Gifar attacks. Upon testing by installing JRE 6 Update 13 on a Windows XP Dell Latitude D630 Laptop, with 2G Ram and a Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz processor, we found the patch to be ineffective against the attacks. Further, this patch does not solve the general issue of content repurposing attacks and is directed only at attacks which affect the Sun's Java Plugins.

Finally, AjaxScope [19], BrowserShield [24], and CoreScript [31] secure the browsers by rewriting HTML and JavaScript. They convert any embedded scripts into safe equivalents by placing filters at run time to protect against known attacks. While this approach could be adapted so as to include some content repurposing attacks in the list of attacks checked for, it still cannot monitor, detect or prevent the actual attacks on the end-user's machine once an attack is launched.

7 Discussion and Concluding Remarks

In this paper we presented a light-weight and effective tool to protect against an emerging class of attacks, namely, content repurposing attacks. This latent threat is poorly if at all addressed by current protection systems, both at the remote sites and locally by antivirus and antispyware. We designed and developed the DeCore, which tool promptly detects a number of possible content repurposing attacks and adds no overhead to the users local machine or browser where it resides. It also does not invasively monitor all the user interactions with the browser. Further, the DeCore effectively stops any ongoing attack. Next, we will improve the accuracy of our system's detection, by enabling detection for subtle attacks. Currently, we cannot determine whether the malicious applet is trying to steal information from password files or whether it is simply scanning the local machine's file system. We are exploring how to supplement the add-on to detect additional attacks by adding more JavaScript based components.

Acknowledgements The work reported in this paper has been partially supported by the NSF grant CNS 08-31247 (2008-2012).

References

1. A. Askarov and A. Sabelfeld. Secure implementation of cryptographic protocols: A case study of mutual distrust. In *ESORICS, LNCS 3679,*. Springer -Verlag, 2005.
2. R. Auger. et al. Threat classification - denial of service. http://www.Webappsec.org/projects/threat/classes/denial_ of_ service.shtml
3. M. Bailey Foreground Security.Superior Security. Visible Results - Flash Origin Policy Issues http://foregroundsecurity.com/MyBlog/flash-origin-policy-issues.html
4. R. Brandis. Exploring below the surface of the gifar iceberg. *Whitepaper*, February 2009.

5. CNET. Cnet Antivirus Software. http://download.cnet.com/windows/antivirus-software/?sort=editorsRating+asc&tag=mncol;pm
6. CNET. Top 10 Anti Spyware Software http://www.top10list.com/top,10,spyware,software/top-ten-spyware-protection.asp
7. M. Dhawan, V. Ganapathy Analyzing Information Flow in JavaScript-based Browser Extensions. In *ACSAC '09: Proceedings of the 2009 Annual Computer Security Applications Conference*, December 2009.
8. Document object model (dom) level 2 events specification. W3C Specifications, November 2000. http://www.w3.org/TR/DOM-Level-2-Events/
9. J. Grossman. Top ten Web hacking techniques of 2008 (official), February 2009.
10. G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *15th Annual Network and Distributed System Security Symposium (NDSS'08)*. February 2008.
11. A. Guha, S. Krishnamurthi, and T. Jim. Using static analysis for ajax intrusion detection. In *WWW '09: Proceedings of the 18th international conference on World wide Web*, ACM. 2009.
12. V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing. In *Third virtual Machine Research and Technology Symposium. USENIX*, 2004.
13. B. Hicks, K. Ahmadizadeh, and P. McDaniel. From languages to systems: Understanding practical application development in security-typed languages. In *22nd Annual Computer Security Applications Conference*, 2006.
14. Inferno's blog on application security. Easy server side fix for the gifar security issue, January 2009. http://securethoughts.com/2009/01/easy-server-side-fix-for-the-gifar-security-issue/
15. B. .E. John, A. Vera, M. Matessa, M. Freed and R. Remington. Automating CPM-Goms. In *Computing Human Interaction*, 2002.
16. C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th ACM World Wide Web Conference*, 2006.
17. C. Karlof, U. Shanka, J. D. Tygar, and D. Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *14th ACM Conference on Computer and Communications Security*, 2007.
18. G. Keizer. Typical Windows user patches every 5 days Computer World. http://www.computerworld.com/s/article/9165738/Typical_Windows_user_patches_every_5_days
19. E. Kiciman and B. Livshits. Ajaxscope: A platform for remotely monitoring the client-side behavior of Web 2.0 applications. In *ACM SOSP Symposium on Operating Systems Principles*, 2007.
20. L. MacVittie. The Web 2.0 botnet: Twisting twitter and automated collaboration. http://devcentral.f5.com/Weblogs/macvittie/archive/2009/04/13/the-Web-2.0-botnet-twisting-twitter-and-automated-collaboration.aspx
21. J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer. Shamon: A system for distributed mandatory access control. In *Computer Security Applications Conference*, 2006.
22. E. Mills. Cnet news. Researchers warn of malware hidden in .zip files http://news.cnet.com/8301-27080_3-20002542-245.html. April, 2010.
23. nsIFile - Mozilla development center. Developer's Guide, May 2009.
24. C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. In *USENIX OSDI Symposium on Operating Systems Design and Implementation*, 2006.
25. B. Rios. Billy (bk) Rios, Thoughts on security in an uncivilized world. Blog. http://xs-sniper.com/blog/ Last Accessed: February, 2010.
26. F. E. Ritter, G. J. Baxter, G. Jones, and R. M. Young. Supporting cognitive models as users In *ACM Transactions on Computer-Human Interaction, 7,* 2000.

27. G. J. Sharif M., Singh K. and L. W. Understanding precision in host based intrusion detection. In *RAID Recent Advances in Intrusion Detection*, 2007.
28. B. E. Ur and V. Ganapathy. Evaluating attack amplification in online social networks. In *W2SP'09: 2009 Web 2.0 Security and Privacy Workshop*, May 2009.
29. P. Wayner Mimic Functions. *Cryptologia XVI(3)*. 1992.
30. P. Wayner Disappearing cryptography *3rd Edition: information hiding: steganography & watermarking*. MK/Morgan Kaufmann Publishers. 2009.
31. D. Yu, A. Chander, N. Islam and I. Serikov. JavaScript instrumentation for browser security. *In ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2007.