Image Repurposing for Gifar-Based Attacks

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

ABSTRACT

In this paper we present a light-weight and effective tool to protect against a type of repurposing attack vector, which exploits weaknesses in Web applications, referred to as the Gifar. A Gifar-based attack relies on 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. In the paper, we explore the capabilities of Gifar-based attacks, expose the efficacy and the state of the art of the existing protection mechanisms against these type of attacks. We show that despite some initial steps taken by many Web portals to prevent from similar attacks, Gifar attacks can be launched from any sites, and can be used as carriers of a variety of disruptive attacks, such as denial of service, command and control, and data theft. We present the *AntiGifar*, a lightweight system that addresses this deficiency by detecting and stopping these attacks at the client end. As demonstrated by our test results, our solution promptly detects a number of possible Gifar-based attacks launched by any Web site. The *AntiGifar* provides an effective solution to this subtle threat while adding no overhead to the users local machine or browser where it resides, and it does not invasively monitor the user interactions with the browser.

1. INTRODUCTION

The emergence of Web 2.0 has brought with 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. The popularity of these applications and the Web-based communities, combined with their ability to allow users to upload their own content has opened ways for new attacks on the browsers such as HTTP response splitting attacks. These social computing platforms are nowadays an easy target of hackers and phishers alike, to whom the user content represents a wealth of user information. The Web 2.0 open availability of data combined with the sheer amount of personal information shared on these platforms, such as in Social Network sites, further worsens the impact of any attacks aimed at stealing user information.

On these platforms, users can upload content which may potentially contain 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 remotely stored users' 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. 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 [25], different forms of distributed denial of service attacks [37], [3] and various forms of malware exploring the internal structure of the Web 2.0 platform.

In this work, we focus on a specific example of repurposing attacks. This attack vector, first introduced by Billy Rios and Petko D. Petkov [5], 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. Up until recently, Gifars have been cited as a leading threat [14]. In order to ground our work, we thoroughly investigate the Gifar-based attack in all its facets: the possible forms this attack can take, the actual disruptive power in real-world setting, and the existing mechanisms currently deployed to protect from similar attacks. To prove the effectiveness of using the Gifar as an attack vector, we present a slew of attacks which can be launched easily using Gifars. These attacks help us demonstrate the ability of the Gifar to manipulate and steal information from the local machine of the victim, when the Gifar files are opened in the victim's browser. Despite the Gifars being such powerful attack vectors, our experimental evaluation shows that the existing users' defense mechanisms do not recognize the Gifar files as malicious, nor do they raise an alarm when the attacks crafted by us are actually being executed. Our tests also demonstrate that the Gifars can be uploaded to numerous popular Web sites, including Picasa [30] , Orkut [29] and Friendster [13]. Surprisingly, even common antivirus or antispyware fail in detecting an ongoing attack. The reason behind the power of this relative simple attack vector lies in its ability to masquerade its actions as ordinary browsers' operations: the operations performed by the Gifars when they are loaded in the browsers are often the same type of operations needed by the Web applications to genuinely perform their tasks.

In light of our findings, we build a protection mechanism, referred to as *AntiGifar* specific to Gifar-based attacks. We choose a client-end architecture rather than a server-side solution. The client side solution affords better protection to the user because it effectively allows us to monitor the user's interactions with the browser without invasively monitoring the specifics of the input. Further, as confirmed by our preliminary tests conducted on existing platforms, if the protection is at the server-end, the attacker can overcome server-based protection by hosting the Gifar on some other remote, malicious server and launch an attack on the end user's system by tricking the user into click the link which launches the applet in the Gifar.

We design the AntiGifar building on the notion of *control-flow* *graph*, which models interactions of the end-user's machine and browser with the Web site, and helps detecting possible anomalies. 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. 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.

As a proof-of-concept, we deploy the AntiGifar as an add-on for Mozilla Firefox, using JavaScript. The system is a lightweight, yet accurate protection mechanism, which protects the end user from various types of Gifar-based attacks. As demonstrated by our test results, the system 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. Though certain limitations exist, this is one of the first effective solutions against Gifar-based attacks.

The main contributions for this paper are summarized as follows:

- We test various Gifar-based attacks and their effectiveness by deploying Gifars on real-world Web-sites.
- We develop the notion of control-flow graph, to model the normal behavior of a user and her browser while interacting with a Web site.
- We draw on the control-flow graph to develop a lightweight protection mechanism against Gifar-based attacks.
- We prove through extensive tests that the developed system is accurate, robust and resilient to different types of Gifar-based attacks.

The rest of the paper is organized as follows. In the next section, we elaborate on the Gifar attack, and discuss its applicability in existing Web sites. As part of our analysis, we explore existing protection mechanisms, and highlight their limitations. In Section 3 we present the design of the AntiGifar followed by the system's deployment and testing. We discuss related works in Section 5 and conclude in Section 6.

2. GIFAR-BASED ATTACKS

In this section we provide some background information about Gifar-based attacks. We begin with describing the attack vector features, following with the results of our preliminary investigation on real-world settings and on a social network site developed by us. Then, we discuss the protection mechanisms currently applied to protect from similar attacks.

2.1 The Gifar-Based Attack

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 result is a flexible and powerful attack vector, that attackers can exploit by uploading such Gifars to Web portals which host these images from their domain. 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. The browser then opens the image containing the applet as a JAR and executes the code in it.

```
<!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>
```

Figure 1: Example HTML code used to invoke an applet in a browser

An example of the HTML file including the Gifar file is reported in Figure 1. As shown, 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.

The Gifar image can be loaded to a Web site's photo application or it could be sent to a user as part of an HTML based message. In order to send the file as a part of an HTML based message, the Gifar is usually hosted on some remote server and the HTML message then embeds the Gifar as an image file using the $\langle img... \rangle$ tags. The embedded image can then be linked to the Web page which invokes the applet, as described above, using the $\langle ahref... \rangle$ tags. When the victim clicks on the HTML based message, the applet is invoked as the Web page invoking the Gifar as an applet is opened.

Depending on what code is embedded and where the applet is being run from, the attacker can exploit the running applet in several ways, from gaining or alter data remotely stored (e.g., in the user's profile on a social network site), to collect and modify data in the user's local system, choking a victim's browser, or even flooding the victim's Web space. Examples showing the potential of Gifar-based attacks are:

- *Remote Intrusion*: By using a Gifar, the attacker can open an explorer window which allows him to explore the victim's machine from his remote location. 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.

- *Denial of Service or choke a victim's system*: The JAR file included in this Gifar is designed to launch 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. This attack can further be modified into a DDoS attack. The Web server hosting the targeted site, can be subjected to a DDoS if the attacker posts a number of Gifars 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.

- *Command and Control attacks*: The Gifars could provide the attacker a distributed command and control channel for the botnets owned by the attacker. The command and control channels are used by the attacker to remotely control the botnets [15]. The attackers can easily create and embed their server and/or client programs as the JARS, such that once the HTML page invoking the applet embedded in the Gifar is loaded, botnets receive their commands and begin to carry out the malicious operations. The Gifars make good attack vectors for carrying out Command and Control attacks since theses images can be easily distributed by loading them on public portals, such as social networks, virtual communities, blogs etc.

- *Bypass the authentication*: In order to bypass the control of certain authentication-protected Web sites, the user will have to download the JAR file which completes the attack. This JAR file can be delivered using a Gifar image. This attack proceeds by retrieving the saved cookies of the user and using them for login. There are two programs running, one which extracts and forwards the cookies to the attacker and one which uses them for login. 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 the SN. 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.

  Both the command and control attacks and a form of denial of service attacks can be carried out by modifying system files. The attacker could modify the password files of the victim, causing the authentication to fail repeatedly, when the victim tries to log in using a saved password. The attacker can additionally issue the commands in the command and control attacks by modifying files stored on the victim's local machine.

2.2 Testing the Gifar in real-world settings

In order to assess the potential of Gifar-based attacks, we have extensively tested the examples discussed in the previous section. To conduct our tests, we used a two-fold approach: first, we tried all the discussed attacks in a harmless fashion on popular real-world content management or social network sites, to check whether any protection mechanism is currently in place for this attack. Second, we extensively tested the attack using a local Web server, hosting an open source content management system created by us. Using such a methodology we could evaluate both the feasibility of the attacks as well as their effective disruptive power. Although there are differences in each Web 2.0 platform that lead to slightly different conclusions, our analysis is thorough enough to ensure a good level of generality. Our tests show that Gifar-based attacks are the result of a number of vulnerabilities. First, the Java Runtime Environment (JRE) does not check the extension of files before parsing the JARs. Second, 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. Ideally, the fact that a file with an image extension such as GIF or JPEG is being opened as a JAR executable should signal a suspicious activity. However, none of the existing browsers raises such flags. 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. Finally, Web applications, such as those that manage pictures, often take control of user files and store them remotely. This gives the embedded applet access to all the files and records which are available to the Web application.

We now provide some interesting observations resulting from our evaluation. On Orkut [29] and on Friendster[13], the Gifar cannot be directly uploaded to the photos. However, the Gifar can still be posted by embedding the images using the sites' public message systems (called *scraps* on Orkut). Further, these scraps can be linked to the page which is used for launching the applets embedded in the Gifar and therefore the attack itself can be triggered by a victim simply clicking on a comment posted to her account.

In case of Picasa[30], the Picasa server allows us to load the Gifar images directly using the photo loading interface. These Gifar images are also saved without any modification or loss of information and without the embedded JARs being corrupted whenever any user saves the Gifars onto their system. Further, since Picasa serves the Gifar from its Web domain, lh4.ggpht.com, which is in fact the same as the Google Web domain, lh4.google.com [32], we can directly use the victim's Web cookies whenever he is signed into Google. This allows the attacker to bypass the authentication even more easily as the attacker does not need to steal the cookies or the passwords. Finally, the "remember me" option does not need to be enabled for the attacker to be able to bypass the authentication, since the attacker can use the victim's cookies if the victim is logged into the Google domain.

Among others, we tried Gifar-based attacks in case of LiveJournal [24] and the art community DeviantArt [7]. Both the sites allow us to load the Gifar, and similar to Picasa, the Gifars are stored without modification. The attack can therefore be launched in a similar method to the way it is launched on Picasa. Further, we can also use the method described above for Orkut and Friendster, and post the Gifar in messages and linking them to the malicious site used to trigger the applets in the Gifars.

Facebook[10] has proved to be more resilient, as compared to other social network sites. Facebook actually scrubs all the uploaded pictures, removing any possible embedded code. The image visible on the site is not the one which is uploaded by the user, rather, it is a resized version of the original. Furthermore, Facebook does not allow the posting of HTML code in the network, although it allows HTML links to be posted. The possibility of adding links opens a potential breach: Facebook can be used as a site to lure the unsuspecting victims to launching Gifars which may have been copied to their local machines. Hence, despite these precautions, even Facebook is not immune from Gifar-based attacks. The attacks for example can be launched using its application "Advanced Wall" which allows us to post HTML content. Further, other applications could be used to distribute the Gifars for the attacks, since these applications make it possible to load and retrieve the original Gifar.

As introduced, we tested the attacks built by us on a social network (SN) site using Drupal [9], which is an open source content management system. Drupal is widely used by smaller corporations to host Web sites and intranets. On SNs built using Drupal, all these attacks are very easily launched since the Drupal based SNs allow the attackers to load a Gifar directly. Further, these images are not scrubbed and Drupal based SNs often make the original Gifar visible to the users on their Web site. The attacker can therefore easily locate the JAR files and launch these files from an HTML file. We tested all the four attacks detailed above with Drupal. All the attacks succeed, in almost no time. In case of the bypass authentication attack, since the files being copied are very small, the time taken for such copying is about 16 milliseconds. Also, we observed that since the images are served from the Drupal server's domain, it is really easy to bypass the authentication when the victim is logged in to the Drupal system. Further, the JAR file has access to all the other files stored on the server as the actual Gifar is stored in the same domain as the remainder of server files, making it easy to modify these files and steal the victim's information.

Finally, the Gifar-based attacks are also not identified by the popular AntiSpyware and Antivirus solutions which are available to users. We tested the top 5 Antiviruses and the top 5 AntiSpyware [1] as listed by CNet [39, 38], 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 the Gifar-based 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 Files` are modified whenever the user changes a password or asks the browser to remember another password. The AntiSpyware do not recognize the Gifar-based 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.

2.3 Existing protection mechanisms

Current systems try to protect from Java-based attacks in different manners, 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 defense currently implemented.

- Using a "throwaway" server for images: Using a "throwaway" server means that a different server hosts the images which are actually viewable. The original images are never presented to the users, ensuring that the attacker cannot find the path to the original file to invoke the Java applet. This thwarts any Gifar-based attack. Though this solution can be very attractive to large corporations which are dedicated to running highly popular portals, smaller portals cannot adopt this approach due to the cost of utilizing another server to manage the front end. Further, the attack code, embedded in the Gifar still exists on the remote site, leaving a vulnerability waiting to be exploited.

- Ensuring that only authenticated scripts can run in the server space: Before any script searches the database or runs in the server space, it is required that the script presents some form of authentication to the server or some trusted entity. This potentially prevents unauthenticated scripts from gathering user information. However, it does not prevent authenticated scripts from leaking information they gathered legitimately. Further, it does not prevent the Gifars 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.

[1]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.

- 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. Resizing images often causes the embedded code in the Gifar to be corrupted or lost. While filtering 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 solutions, which have been considered to block these types of attack vectors [32], [19]. For example, a user who avoids the use of persistent cookies can help prevent an attacker from bypassing Web-sites authentication. This approach, however, cannot be deemed as a practical solution because of the popularity of such persistent cookies.

On the server side, the Web portals can help avoid the attack of bypassing the authentication by checking whether a persistent cookie comes from the same machine, it was previously used. If not, it can ask for further verification from the user, using security questions. This process may be viewed as cumbersome by many users. Further, an attacker can always spoof the victim's IP address in order to gain authentication using the stolen passwords or the saved cookies of the victim. Web portals could also opt for limiting/blocking the HTML links posted on their sites. This, however, may not be a well-accepted solution since the ability to post arbitrary comments contributes largely to the popularity of many content management portals.

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 to stop the attacks. 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 ANTIGIFAR SYSTEM

In order to protect from Gifar attacks, we have designed a lightweight protection mechanism, referred to as *The AntiGifar*. Using the attacks described in Section 2.1 as representative attack types, our aim is to create a general protection mechanism for both the victim's local system and her remote data. The AntiGifar can be successfully deployed at the user-end, in the form of a simple browser plug-in, or as a component at remote server. Since most of the Gifar attacks aim at attacking resources on a end-user's machine, the client end constitutes an important point of protection. The user will be able to control the attacks, and customize the protection mechanism as needed based on his/her browsing history. On the other hand, if deployed as a remote server component, the solution will be transparent to end users, while providing an effective approach against Gifar-based attacks. While a server-based solution

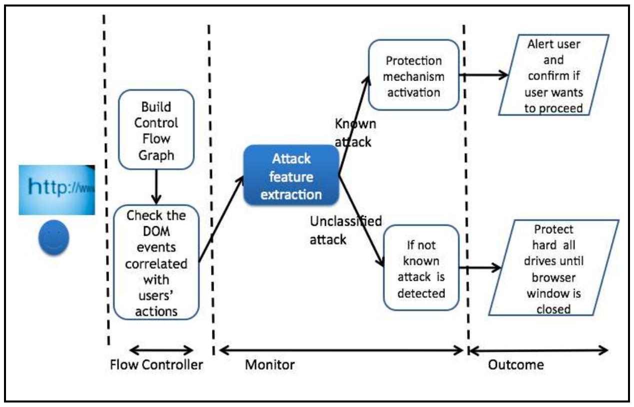

Build Control Flow Graph
Check the DOM events correlated with users' actions
Attack feature extraction
Known attack
Unclassified attack
Protection mechanism activation
If not known attack is detected
Alert user and confirm if user wants to proceed
Protect hard all drives until browser window is closed
Flow Controller
Monitor
Outcome

Figure 2: The AntiGifar system's logic flow.

is effective in detecting attacks that target the server itself as well as attacks that aim at using the server as an attack vector, it is not able to protect from attacks that lure the user into logging into different realms. Above all, the very nature of these attacks exploit a basic vulnerability of the browser and various plugins which are downloaded and managed by the client. Therefore, a client-side solution is arguably the most natural approach for handling such vulnerabilities.

The AntiGifar main execution steps are sketched in Figure 2. The depicted tasks are carried out by two logical components: the *flow monitor*, and the *AntiGifar detector* (detector, for short). These two components are discussed in the following sections.

3.1 The AntiGifar flow monitor

The AntiGifar is designed in the form of a light-weight monitor at the client-end, that stops attacks as it detects them. Hence, the first tasks toward Gifar-attacks detection consist of monitoring whether an attack is ongoing. Such tasks are carried out by the *flow monitor* component. The flow monitor is built based on the observation that any given Web site follows a specific information flow, from a user's end system to a remote server, and possibly stores users' data in Web server's database systems. The Gifar-based attacks violate these information flow rules as they allow for illegitimate transfer of information between one domain to another. Hence, our approach is to detect such violations in an efficient and accurate manner.

This requires accomplishing two main tasks: (i) capturing all the interactions between the user and the browser (ii) 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.

To carry out task (i), the flow monitor 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 CFG is derived by considering all the possible DOM events and JavaScript links when looking at the source code of a Web-site.

EXAMPLE 3.1. *Figure 3 provides an example of a control-flow graph between a SN, the user's local system and an external Web site's domain. In the figure, the edges F0, F1, F2, F3, F5, F8, F10 and F11 denote legitimate flows. A crossed edge between the comments or messages, and any entity indicates a malicious operation being performed. In our example, these correspond to edges F4, F6, F7, F9, F12 and F13.*

For example, the edge F0 represents a password file being read by a social network site and verified by the social network's database when the user clicks on the login button, while edge F1 represents the display of the Web site caused after the user has logged in. The edges F2 and F3 represent the user's profile being updated, which are caused when the user clicks the links and submits the required forms. Edges F8, F10 and F11 represent update and read actions on the external Web site.

The edge F5 signals a new Web site being opened as a result of the user's click on the social network site. A scrap 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 scrap using any I/O operation (F13), it signals that an external entity is trying to control something on the social network site.

Once the graph is loaded, the flow monitor carries out task (ii), by verifying that 1) the information flows from and to legitimate states as prescribed by the graph, and 2) 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.

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, 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.

Listening to DOM events is effective since Gifar-based attacks are carried out by completing a few seemingly normal events such as redirection to an external Web site from a source site, reading of the password files etc. (these events are DOM events themselves). The fact that these DOM events are not caused by the user's interaction or input is what sets them apart from the legitimate events which are needed to reach the legitimate state.

EXAMPLE 3.2. *In the control-flow graph introduced in Example 3.1, the paths traversed to reach these states indicate the difference between an ongoing attack and a normal operation. For example, in Figure 3, both the edges labeled F8 and F9 represent a flow from the state $ExternalWebsiteDisplay$ i.e $s4$ to the state $ExternalDBWrite$ i.e $s5$, the edge F8, which is a legitimate flow, has the DOM operation* click *associated with it, while the edge F9, which denotes an illegitimate flow, does not. The requirement that a click leads from the state s_4 to the state s_5 differentiates the legitimate flow from an illegitimate flow.*

Further, to improve detection accuracy, the monitor not only relies on correlating DOM events, but also 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 a level of sophistication that is unlikely. Such simulation requires the use of sophisticated HCI models such as GOMS and UIMs besides a huge database of similar activity by human beings [20, 33]. Finally, in order to minimize possible attempts of this type, the system requests feedback from the end user upon detecting an attack. For example, the AntiGifar notifies the user when a Gifar attack is suspected. These methods are discussed in detail below.

3.2 The AntiGifar Detector

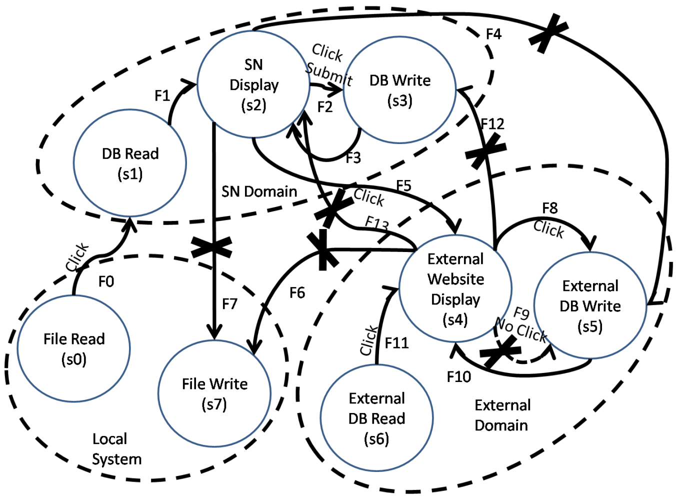


Figure 3: Example of Control-Flow graph between an external Web site's domain, a social network the domain and the end-user's local domain

While the FSM detects a potential Gifar-based attack, it does not identify the specific operation carried out by the embedded Jar file. This task lies with the detector. The detector is in charge of classifying the suspected attack, and of taking subsequent proper actions. The actions taken upon detection vary according to the user-specified degree of protection: the user can configure the AntiGifar to either send alerts as soon as a Gifar attack is suspected, or to actively stop the attack whenever detected, by immediately closing the window hosting the Gifar.

The detector has 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 user. First, the most disruptive and quickest possible attacks are checked. That is, first the system checks whether the Gifar aims to modify the user's files. Using this attack, the attacker may change the user's privacy preferences and passwords thus opening the door for further destructive attacks. If this attack is not under way then it checks for progressively lesser disruptive attacks.

Precisely, the monitor uses JavaScript to zero in on the particular attacks in the below order.

Modification of a local file on the system and Command and Control attacks: A JavaScript based component is used to check whether any of the user's system files are being modified by using the command file using `file.lastmodifiedtime`. The AntiGifar accesses the files using `nsIFile` functions. 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 [2]. The detector uses such command to verify periodically that any change in the user's system is either correlated by a DOM event or is a system event, and whether any modification or file creation is generated by a user-controlled application. For example, the detector compares the time the system's password file (e.g., signons3.txt for Mozilla Firefox) was last modified with user related DOM events such as loading of a form page which allows resetting of passwords and mouse click. This type of check is performed periodically, until the Web page hosting the suspected Gifar is not closed. In the example of the password, if the user submits any changes on the form page, then only should the file be modified. If the file modified is not correlated to the user submitting a password form, then the attack is said to be ongoing. Similarly, if files are downloaded by the Internet and they are from the site which is suspected to host the Gifar, and not correlated by 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.

Modification of the user's remote profile by code running on the Web site: The Gifar can easily target Web sites that host some user-generated content, such as social networks, and blogs. Once installed, the Gifar can, for example, add spamming content, malicious links or modify the user posted content. To address this issue, 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. 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 match the same 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 AntiGifar at the client-end. To limit the scope of the monitoring, the user can create a list of such sites that the AntiGifar 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.

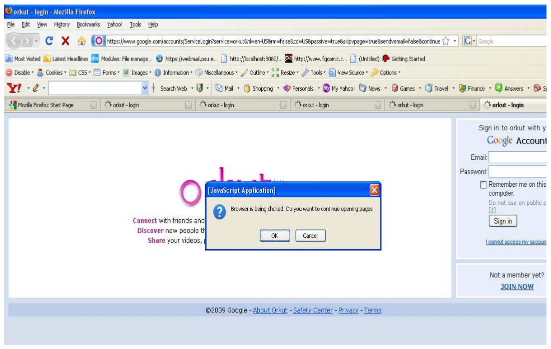
Choking attacks: The JavaScript component dealing with this attack listens for page load events and mouse click events. To avoid false positives, the detector checks whether the number of mouse click events are not only the same as the number of page events, but also that they were executed at the same 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.The user is then alerted to a possible attack and asked whether he wants to continue opening multiple pages. He is also asked to close the Web site hosting the suspicious Gifar.

A screenshot showing how the attack is stopped is reported in Figure 4. The user is being asked to confirm whether he/she wants to continue with opening the windows when 4 tabs of the same page were opened without any action on the user's part[3].

Other subtle attacks, such as attacks aimed at bypassing the authentication which do not result in the change of state of any of the documents or files, are handled as follows. Upon suspecting a pos-

[2]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 [28]

[3]For testing purposes, a small number of windows (four windows) were allowed to be opened without the user's action before the confirmation is requested for the purpose of simulation of the attack, but this number can be increased or decreased very easily.


[JavaScript Application]

Browser is being choked. Do you want to continue opening pages

orkut - login - Mozilla Firefox
OK
Cancel
Sign in
JOIN NOW
©2009 Google - About Orkut - Safety Center - Privacy - Terms

Figure 4: The AntiGifar's Screenshot for the Choking Attack.

sible attack, the AntiGifar caches copies of the all the changes on the files modified while the attack is deemed undergoing, so that no permanent modification is performed on the client machine. The user is asked for confirmation regarding the data being rendered until the malicious activity is ended. Hence, the user is informed of any files' change, before committing them. The file system integrity is monitored by matching hash values calculated before and after the attack is suspected.

Notice that the JavaScript components used by AntiGifar 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 AntiGifar covers the class of attacks where a victim's browser is rendered useless to him as it is taken over by a malicious script.

4. THE ANTIGIFAR EVALUATION

We developed the AntiGifar as an add-on for Mozilla firefox version 3.0.10. Most of the add-on is written in JavaScript or uses Jar files, hence it is easy to port this solution to any browser. All references to the files and file paths are not platform dependent, thus making it compatible with different platforms and file systems. To port the AntiGifar onto a different browser or OS, the files references must be configured according to the chosen platform. For example, in order to port the AntiGifar to Internet Explorer, the JavaScript component of the monitor must check the `Microsoft Credential file` to ensure the HTTP Authentication Passwords are not tampered with. These files also change with the version of the browser being used [4].

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 overhead added by the Antigifar. Second, we assessed the accuracy of the AntiGifar.

In the first set of tests, we compared the execution time for the Firefox browser with and without the AntiGifar 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. For each run, to ensure accuracy of the experiment, we completed 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 AntiGifar-enabled browser, the same methodology was used, but we first enabled the plugin, at each run. We reported no overhead caused by the AntiGifar 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 be started 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 addon and 46 % without the add-on. The usage for the Firefox session being restored with 6 windows and 60 tabs was 56 % without the addon and 54 % with the addon).

Our second set of tests aimed at verifying the accuracy of the AntiGifar. To this extent, we carried out several different experiments of increasing complexity. First, we begin with assessing false positive rates, i.e. whether the AntiGifar 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 AntiGifar running while 100 different sites were visited to test the accuracy of our system when it is continuously monitoring for Gifar-based attacks. The sites were selected based on their popularity and on the presence of active components. For example, 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 *Glamour*) and blogs.

With our second round of experiments, we increased complexity of the previous test by introducing new Web pages created by us, where the Gifar was carried on pages that hosted other benign JAR files and/or JavaScript components, making it harder to distinguish whether an attack was in place or not. 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. 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 Gifar had some variations in itself, i.e. the source code of the page which invoked the Gifar was different for each try and the page carried different types of JAR files and JavaScript components every time. 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 and even simple games. The AntiGifar proved to be accurate in both set of tests, detecting the attacks correctly.
Finally, in order to further challenge the AntiGifar, we created a new test case by launching multiple attacks at the same time. For this test case, we crafted attacks so as to combine more than one Gifar attack on a single HTML page. We constructed the attacks in two alternative forms: we either hosted multiple Gifars in a same page, or created a JAR file which would carry out different attacks in a single file. With this attack, we were not only interested in checking whether the AntiGifar 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 out of possibly 50 aggregated attacks, 10 times each. The different types of attack were constructed by combining the Gifar attacks discussed in Section 2.1 with one another. We had 4 basic attacks to tests, in possibly 50 ways, calculated as dispo-

[4]`singons3.txt` is used in Firefox 3.0 whereas `signons.txt` is used in versions of Firefox earlier than Firefox 1.5.0.10)

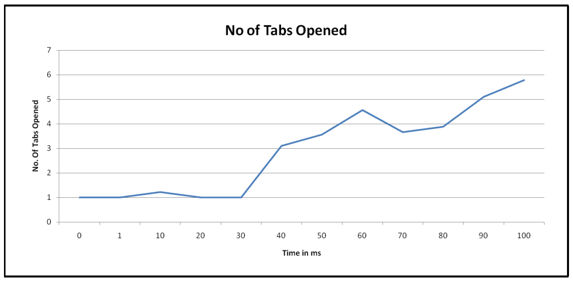


Figure 5: Chocking attack. Evaluation results.

sitions of groups of k from a set of n without repetition, where k can be 2, 3 or 4. Precisely: $c = \frac{4!}{(4-2)!} + \frac{4!}{(4-3)!} + 4!$. We focused on a subset of not trivial combinations, namely five attacks with 2 Gifars hosted at each page, five attacks with 3 Gifars and 2 combinations of all the 4 Gifars, 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 antigifar 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, the discarded combination of attacks). 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 AntiGifar.

In light of this result, we conducted an additional test, and tested the number of pages that could be opened in the choking attack over time by factoring in an artificial delay in our tests from 0 ms to 100 ms. The results of this test are shown in Figure 5. Up to a delay of 35 ms, the average number of tabs opened is approximately just 1. The number of tabs opened starts increasing after 35 ms. Even for a delay of 100 ms, the average no. of tabs opened is only 5.6, which is not enough to slow down the victim's system or crash his browser. Since there is only a very slight chance, if any, of a delay more than 100 ms, we conclude that the attacker cannot cause any significant damage.

5. RELATED WORK

Gifar-based attacks have risen as a new type of web application threats, where images are used to load malicious codes into web applications and to the servers from which these applications run. Gifars have been cited to be the leading threat of 2008 [14] ever since they were discovered by Billy Rios.

An interesting solution specific to Gifar-based attacks, has been proposed in [19]. The idea is to detect the Gifar-based attacks at the server side, so at to detect any Gifar files which are loaded on the server. However, the attack can still be carried out on the victim's browser if the Gifar is hosted by the attacker on a some remote server of his own, since, at that time, the code cannot be used to identify the remote Gifar file being loaded and executed. These remotely hosted Gifar files can be used in the SN to attack a victim as demonstrated by the attacks outlined by us in Section 2.2.

Since to our knowledge, no other solutions for Gifar attacks have ever been proposed, we summarize some of the most closely related approaches recently proposed to thwart generic Java-based attacks. There are two parallel lines of work that are of interest to us: using monitoring-based systems [12], [11] or information flow control strategies [2, 27, 18].

In [16] 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 us in the approach, Guha and colleagues focus on the response to the server from the client and may therefore not be able to detect several of our attacks, since our attacks run at the client end. Further, the proposed solution needs to be run as proxy between the server and the client, which invasively evaluates the response from the client machine. Our solution is much 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 attacker. However, we also plan to enable our solution to detect a wider range of attacks in the future.

A similar approach to the above is taken in [5]by Dhawan et al. The authors develop a system which uses in-browser information tracking to analyze JavaScript extensions. This solution is similar to ours in that the authors study the information flow by considering the DOM events that are used by the JavaSrcipt Extensions to study whether sensitive data is being leaked by them. Despite the similarity, this solution is applicable to only JavaScript Extensions, it does not monitor the malicious behavior of any outside code and also does not detect Java based attacks. We differ from this solution in that we do not have to monitor the sensitivity level of each object submitted by the user over the internet and therefore our monitoring is far less invasive. Further, our solution is better compatible with the existing browsers in that we do not require any modifications to any interpreter of Firefox, unlike this solution, which modifies SpiderMonkey, Firefox's JavaScript interpreter.

Another interesting attempt aimed towards thwarting Java-based attacks is from Soman et al [36]. Soman and colleagues deal with detection of Java-based attacks by analyzing the Java Virtual Machine (JVM). The authors have developed an auditing facility, which audits thread-level changes in the JVM and an intrusion detection tool that uses audit data generated by this facility to detect any Java -based attacks.While this work is geared towards detecting similar attacks as ours, we differ in the approach, in that we do not invasively monitor the clients JVM in order to detect the attacks. Further, Soman's approach is not geared towards specifically detecting the images being re-purposed as attack vectors carrying Java code. Also, Sharif et al. [34] prove that intrusion detection systems that monitor control flow are more precise than system call monitors. The overhead incurred by this method is significantly higher than ours, with performance degradation going up to even 44 %

Since Gifar 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) [23]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 [18]. 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[26], has been often used to regulate the flow of information within the system and the between the processes. A reference monitor is used to enforce Mandatory Access Control (MAC) Policies, which are then used to make security guarantees on a system. With the use of remote attestation and virtual machines [17], 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 for these attacks which is supposed to be applicable for Java Web Start (JWS) and Java Plug-in later than the Sun JDK and JRE 6 Update 10, JDK and JRE 5.0 Update 16 , and SDK and JRE 1.4.2_18. In order to confirm whether the patch successfully prevented Gifar attacks, we tested out the patch 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 then hosted a Gifar on a remote web space and tried to run the applet embedded in the Gifar on the machine. We found that the applet ran successfully despite the update. In order to further confirm the viability of using the patch as a protection for Gifar attacks, we hosted the Gifar on the local machine and launched the same Gifar applet using an HTML file and found that the Gifar applet still worked. We carried out the same tests on three different machines by completely uninstalling Java and the reinstalling the JRE 6 update 16 one one of the machines and by installing JDK 6 Update 16 with Java EE on the other two machines and confirmed that the Gifar applet runs whether it is hosted on a remote site or on the local machine. Each of these machines ran the Windows XP Professional Edition with Service Pack 3 and were Dell Latitude D830 Laptops, with 2G Ram and a Intel(R) Core(TM)2 Duo CPU T7500 @ 2.20GHz processor. This result leads us to believe that the patch is still not fully successful in preventing Java attacks[5].

Finally, AjaxScope [22], BrowserShield [31], and CoreScript [40] 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 Gifar-based 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 a Gifar is launched. It can at most be used to detect that a Gifar attack is running, but not control the effects of the attack on the end-user's machine.

6. DISCUSSION AND CONCLUDING REMARKS

In this paper we presented a light-weight and effective tool to protect against a so far underestimated attack vector, namely, the Gifar. We explored through extensive tests the efficacy of the attacks perpetrated using Gifar files, and their potential.Our analysis leads to the conclusion that 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 also showed the Gifar attacks have the potential of causing serious damages to end users' systems in a variety of manners.

We designed and developed the Antigifar, which addresses this deficiency by detecting these attacks at the client end. The tool promptly detects a number of possible Gifar-based 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 AntiGifar effectively stops any attacks in progress even if it is installed only after a Gifar attack has been launched.
The Gifar is a representative attack of a larger class of attacks, based on repurposing the content. With the AntiGifar we successfully proved the ability to thwart one of such attacks. Therefore, our next step is to extend our protection mechanism for repurposing content attacks. For example, we tested the ability of the AntiGifar to detect recently discovered flash attacks[4]. Our current solution can identify such attacks, so long as they are combined with some other file type and they carry out anyone of the defined attacks. This renders those attacks where other uploadable file types are appended with flash files useless. To improve the accuracy and scope of our approach, we will modify the monitor component, so as to check the HTTP headers.This new control should detect cases when there is no file being explicitly called, like for example the case when the flash files are sent as a mail attachment and opened in the browser.

Further, the AntiGifar itself can be further refined, to improve the accuracy of detection. Currently, the AntiGifar cannot differentiate between various subtle attacks. For example, the AntiGifar 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. To address this limitation, 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).

7. REFERENCES

[1] A. Acquisiti and R. Gross Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook In *Proceedings of 6th Workshop on Privacy Enhancing Technologies*, pages 36-58, Cambridge, U.K, June 28-30 2006.

[2] A. Askarov and A. Sabelfeld. Secure implementation of cryptographic protocols: A case study of mutual distrust. In *ESORICS*. Springer, 2005.

[3] R. Auger. et al. Threat classification - denial of service. http://www.Webappsec.org/projects/threat/classes/denial_of_service.shtml

[4] M. Bailey Foreground Security.Superior Security. Visible Results - Flash Origin Policy Issues http://foregroundsecurity.com/MyBlog/flash-origin-policy-issues.html

[5] R. Brandis. Exploring below the surface of the gifar iceberg. Whitepaper, February 2009.

[6] M. Dhawan, V. Ganapathy Analyzing Information Flow in JavaScript-based Browser Extensions To appear at ACSAC 2009, Honolulu, Hawaii, December 2009.

[7] DeviantArt. I. DeviantArt. http://www.deviantart.com/. Art Community. Last Accessed: June 05, 2009.

[8] Document object model (dom) level 2 events specification. W3C Specifications, November 2000. http://www.w3.org/TR/DOM-Level-2-Events/

[9] The Drupal Platform. http://www.drupal.org. Last Accessed: June 05, 2009.

[10] Facebook. http://www.facebook.com. SN. Last Accessed: June 05, 2009.

[5]The authors have contacted SUN multiple times regarding this, yet at the time of writing we did not receive any feedback.

[11] H. Feng, J. Giffin, Y. Huang, L. W. Jha, S., and B. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 2004. IEEE Computer Society Press, Los Alamitos (2004).

[12] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of Self for Unix Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.

[13] Friendster. http://www.friendster.com. Last Accessed: June 05, 2009.

[14] J. Grossman. Top ten Web hacking techniques of 2008 (official), February 2009.

[15] GU, G., Z. J. and W. LEE. Botsniffer: Detecting botnet command and control channels in network traffic., February 2008.

[16] 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*, Madrid, Spain, 2009. ACM.

[17] 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.

[18] B. Hicks, K. Ahmadizadeh, and P. McDaniel. From languages to systems: Understanding practical application development in security-typed languages. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 153–164, Washington, DC, USA, 2006. IEEE Computer Society.

[19] 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/

[20] B. .E. John, A. Vera, M. Matessa, M. Freed and R. Remington. Automating CPM-Goms. In *CHI*, 2002, 147-154.

[21] 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.

[22] 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.

[23] P. Lee (IBM) Cross-site scripting -Use a custom tag library to encode dynamic content. http://www.ibm.com/developerworks/tivoli/library/s-csscript/

[24] Livejournal. http://www.livejournal.com/. SN. Last accessed June 05, 2009.

[25] 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

[26] 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. ACSAC '06. 22nd Annual*, 2006.

[27] A. C. Myers, N. Nystrom, L. Zheng and S. Zdancewic. Jif: Java + information flow. Software release, July 2001.

[28] nsIFile - Mozilla development center. Developer's Guide, May 2009.

[29] Orkut. http://www.orkut.com SN. Last Accessed: February, 2010.

[30] Picasa http://picasa.google.com/. SN. Last accessed February, 2010.

[31] 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.

[32] B. Rios. Billy (bk) Rios, Thoughts on security in an uncivilized world. Blog. http://xs-sniper.com/blog/ Last Accessed: February, 2010.

[33] 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, 141-173.

[34] G. J. Sharif M., Singh K. and L. W. Understanding precision in host based intrusion detection. In *RAID Recent Advances in Intrusion Detection*, 2007.

[35] A. K. Sood (SecNiche Security) PDF Silent HTTP Form Repurposing Attacks - Web Penetration Testing http://www.secniche.org/papers/SNS_09_03_PDF_Silent_Form_Re_Purp_Attack.pdf, May 2, 2009.

[36] S. Soman, C. Krintz, and G. Vigna. Detecting malicious java code using virtual machine auditing. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 1–11, Berkeley, CA, USA, 2003. USENIX Association.

[37] B. E. Ur and V. Ganapathy. Evaluating attack amplification in online social networks. In *W2SP'09: 2009 Web 2.0 Security and Privacy Workshop*, Oakland, California, May 2009.

[38] R. Vamosi. Cnet 2007 antivirus performance test scores, 2006.

[39] R. Vamosi. Cnet top 10 antispyware apps, August 2008.

[40] 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.