OSNGuard: Detecting Worms with User Interaction Traces in Online Social Networks

Liang He, Dengguo Feng, Purui Su, Lingyun Ying, Yi Yang, Huafeng Huang, and Huipeng Fang

Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
{heliang,feng,supurui,yly,yangyi,huanghuafeng,fanghuipeng}@tca.iscas.ac.cn

Abstract. In the last few years we have witnessed an incredible development of online social networks (OSNs), which unfortunately causes new security threats, e.g., OSN worms. Different from traditional worms relying on software vulnerabilities, these new worms are able to exploit trust between friends in OSNs. In this paper, a new worm propagation model was proposed, named EP-Model, to find out the common characteristics of OSN worms including XSS-based JavaScript worms and Social-Engineering-based Executable worms. And then we designed OSNGuard, a client-side defense mechanism which could prevent the propagation of OSN worms conforming to the EP-Model. Particularly, starting from tracing relevant user interactions with client processes visiting OSNs, our system could identify and block malicious payload-submissions from worms by analyzing these traced user activities. To prove the effectiveness of OSNGuard, we presented a prototype implementation for Microsoft Windows platform and evaluated it on a small-scale OSN website. The system evaluations showed that OSNGuard could sufficiently protect users against OSN worms in a real-time manner and the performance tests also revealed that our system introduced less than 2.5% memory overhead when simultaneously monitoring up to 10 processes.

Keywords: Worm Detection, Online Social Networks, User Interaction Trace.

1 Introduction

Since its first appearance in 2005, OSN worm has become one of the most serious security issues on the Internet. Although the purpose of Samy, the first OSN worm, was just to propagate across the MySpace without any malicious payload [1], it did infect more than one million users within 20 hours, surpassing traditional worms such as Code Red, Slammer and Blaster [2]. Now numerous new OSN worms released for the purpose of getting privacy and profit have emerged in popular OSN websites, e.g., Twitter and Facebook. In fact, at the time of writing this article, a new OSN worm, named Ohaa, is spreading in Twitter by posting a message containing a shortened malicious URL on user's behalf: "Ohaa habere bak :O goo.gl/VbpzM".

S. Qing et al. (Eds.): ICICS 2013, LNCS 8233, pp. 59–74, 2013.

Owing to the popularity of Social-Engineering-based infection methods adopted by the new worms, conventional countermeasure methods [3,4] which mainly rely on the successful detection of the unique scanning patterns all become insufficient. Furthermore, systems [5,6] that focus on traffic anomaly detection also fail to protect users in that they can find no anomaly but only ordinary HTTP stream data.

Considering the ineffectiveness of traditional methods and the severity of impact of these new worms, the security community has proposed some preliminary solutions to mitigate the new threats. PathCutter [7], with view separation and request authentication, does better in JavaScript worm prevention than Spectator [8], which is the first server-side solution, and Sun's first pure client-side system [9], which is based on string comparison. Unfortunately, as the authors have mentioned, their systems are incapable of preventing the drive-by download executable worms. While Xu proposed an early warning OSN worm detection system [10] which is based on the deployment of decoy nodes collecting the malicious messages, their server-side system works only if the infection rate exceeds an empirical threshold which will cause an inevitable delay.

The aforementioned analysis naturally leads to the questions: *Can we propose a generic approach to effectively prevent current (or future) forms of OSN worms? If yes, how can we prevent them in real-time manner?* In this paper, we address these questions through a detailed study of OSN worms and their propagation model. First, we classify the worms into JavaScript worm (J-worm) and Executable worm (E-worm). Next, we propose a novel worm propagation model, named EP-Model which can describe these new worms. Finally, we find out and extract a common characteristic that can be used to detect and block OSN worms effectively. Sepcifically, we find that although different worms adopt various enticements to trick users, they propagate by submitting payload to servers with forged or without user's confirmations, which means the chance that we can prevent OSN worms if we are able to analyze the relevant user interaction traces to detect those automatic submissions.

In this paper, we propose OSNGuard as a complementary approach that addresses some of the limitations of existing systems. Particularly, OSNGuard correlates submissions from a client with relevant user interaction traces to identify propagation of OSN worms. To this end, OSNGuard introduce several functional modules: *Social Traffic Monitor*, *Social Interaction Tracer* and *OSN Worm Detector*. And to estimate our system, we further implement a system prototype for Windows platform and conduct experimental evaluations on a real but small-scale OSN website built on Elgg [11]. The evaluation results demonstrate that OSNGuard can effectively prevent OSN worms in real-time manner. Moreover, the performance test reveals that delays and memory overhead introduced by OSNGuard can be negligible.

The remainder of this paper is organized as follows. We first discuss classification and motivation in Section 2. Section 3 introduces our OSNGuard system and its components. We present the experimental evaluation results in Section 4. Section 5 discusses some limitations and potential future work. In Section 6 and Section 7, we discuss related work and conclude this paper.

2 Classification and Motivation

2.1 Worm Classification

By their existence forms and infection manners, OSN worms can be classified into two categories, XSS-based JavaScript worms (J-worms) and Social-Engineering-based Executable worms (E-worms).

J-worms. All J-worms exhibit similar behavior in propagating themselves as they all inject malicious script-based payload into HTML pages in which there exist XSS vulnerabilities. However, the J-worms can also be divided into passive and active according to their enticements. For instance, a passive worm such as the Samy propagates itself only to users who visit a victim's infected page by accident. In contrast, an active worms such as the HelloSamy spread more quickly by tricking users (e.g., posting attractive wall posts) to browse an infected page.

E-worms. In terms of E-worms, they all rely on various social engineering techniques to entice normal users to download and install their copies. However, we are able to classify them by their forms of existence. Particularly, the Koobface is a Windows worm that it is only able to propagate among Windows machines. And the Boonana is a Mac worm that is able to only infect clients running Mac OS. From the Figure 1, we can see that although E-worms are usually OS-specific due to the limitation of executable file formats, they can spread in different OSN websites as there is no need for any XSS vulnerability except trust between friends.

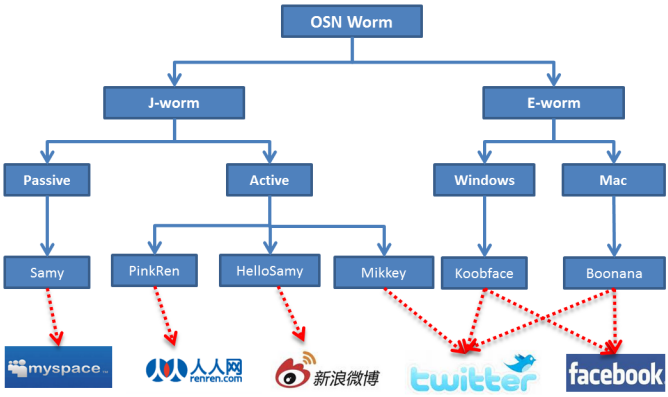


Fig. 1. Classifications of OSN worms and its representative samples

2.2 Case Study

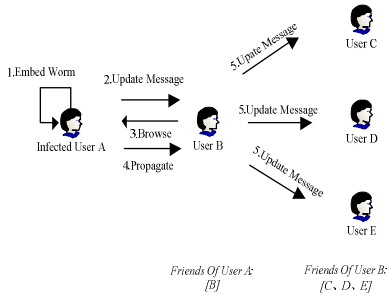
Considering the similarities of propagation methods applied by various worms, in this work, we only present Samy and Koobface as the representative samples for J-worm and E-worm respectively. Actually, based on our bulk analysis of all known worms, for each kind of worms, they all present similar propagation behaviors with Samy (or Koobface). The similarity can also be seen in Cao's survey work [7].

Samy Worm. Figure 2(a) illustrates the propagation of Samy worm. Basically, the worm will first entice users to browse an infected web page in which the worm is embedded (step 1-3). And then when loaded in client browsers, the worm will run to inject itself into victims' pages (step 4-5).

To find out how these J-worms propagate, we also delve into their source code. It is obvious that we can directly analyze the Samy's source code as it is usually embedded into the victims' web pages presented in client browsers. Although there are various forms of existence for J-worms, e.g., Flash or Java Applet, they would ultimately inject the source script code into the HTML pages. Figure 2(b) reveals the pseudo code of the propagation function of Samy who replicates itself by operating an XMLHTTPRequest object to post payload to servers without user's confirmations.

Koobface Worm. Instead of relying on Updating Messages, Koobface will actively send to each of the victim's friends a disguised message usually containing shorten malicious URLs linked to compromised servers in which the worm copies are stored, as shown in Figure 3(a). Based on skilled social engineering techniques and vulnerable trust among friends, the worm is able to infect a large number of users in a very short time.

Similarly, we also intend to analyze the Koobface's source code to chase down how it propagates. However, the Koobface consists of various standalone function modules to accomplish its indispensable tasks, e.g., Downloader, Social Network Propagator, CAPTCHA breaker and Data stealer. As our aim is to prevent its propagation, here we just put our focus on the Propagator which is responsible for sending out worm messages in OSN websites. Figure 3(b) summaries our two-week disassemble analysis result of the executable module which accomplishes its propagation by operating an IWebBrowser2 object offered by Windows platform. From the pseudo code, we can see that Koobface submits the payload with forged user click to propagate itself. Although it may be a simpler method to submit the payload by directly using Socket API functions, we will provide a reasonable explanation why Kobbface dose not do it in this way based on our experiment results provided in Section 4.



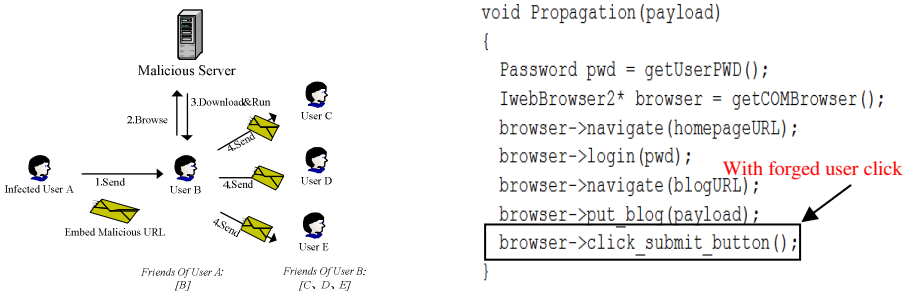
(a) Propagation of Samy

```
function Propagation(payload)
{
  var token = getUserToken();
  var XHR = getXMLHttpRequest();
  XHR.onreadystatechange = function(){
    if(XHR.readyState==4&&XHR.status==200){
      output("Success");
    }
  };
  XHR.open("POST", URL, true);
  XHR.setRequestHeader(attr, token);
  XHR.send(payload);
}
```

Without user confirmation

(b) Pseudo code of propagation

Fig. 2. Details of Samy worm



(a) Propagation of Koobface (b) Pseudo code of propagation

Fig. 3. Details of Koobface worm

2.3 EP-Model and Assumptions

EP-Model. According to the analysis of OSN worms, we propose a new worm propagation model, named EP-Model, as shown in Figure 4. And our goal is to prevent all worms that confirm to this model. Basically, the model is based on the fact that OSN worms propagate themselves with two common sequential steps as follows:

(1) Enticement: Instead of exploiting any software vulnerabilities, OSN worms adopt various enticements to trick users to directly run their payloads. In this sense, these new worms rely on exploiting trust of friends. Specifically, the J-worms usually trick friends to visit one's malicious script-embedded pages and then run themselves in client browsers. For E-worms, they prefer to utilize social engineering-based enticements to trick users to directly download their copies. Once installed in client machines, a worm will run as a single process.

(2) Payload-Submission: To successfully propagate themselves, OSN worms have to try every means to submit their payloads to OSN servers without user's awareness. Based on our aforementioned analysis, the J-worms directly POST their payload without any user's confirmation, whereas the E-worms mimic user's confirmation by firing the button click.

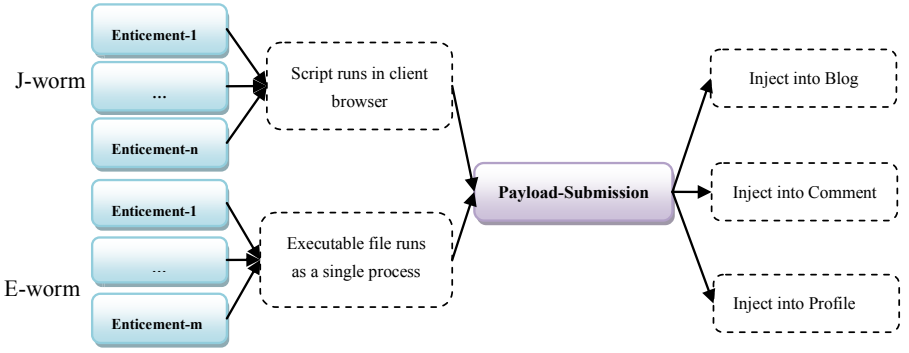


Fig. 4. EP-Model for OSN worms

Assumptions. In this work, all our assumptions are derived from an observed fact that users even equipped with the most advanced anti-viruses can still be infected by OSN worms. So while they are downloaded and run in client machines, we assume that OSN worms should not install any kernel-level rootkits, as otherwise they would have been detected by existing OS-oriented defense mechanisms. Second, we assume the worms do not present any high-risk behaviors in application-level such as adding or modifying Registry entries, tampering with system files and injecting application processes. Obviously, modern anti-viruses can easily notify and prevent these suspicious acts for users. In fact, OSN worms especially for E-worms will run as normal applications to avoid raising any doubt except a popup window for file download request.

2.4 Motivation and Basic Concept

Concerning the existence of overwhelming social engineering technologies nowadays and the vulnerable friendships, there is little work to be deployed to prevent all Enticements from attackers. However, from the EP-Model, we can see that OSN worms share common Payload-Submissions which provide the chance to detect and prevent all of them.

Before our basic idea, we first introduce several relevant concepts used in our following description. Specifically, we refer to any client process connecting OSN websites as a suspicious process. And we use user interaction trace (UIT) to refer to a collected sequence of user actions, such as browse, edit and confirm, which are related to a suspicious process. Finally, we use the term content-submission to broadly refer to submissions conducted by users or OSN worms.

To explain the basic idea, let us consider two real-world scenarios depicted in the top half of Figure 5. Here the Initial Page represents the first web page where a user will meet when she starts to login to the OSN website. Medial Pages are the pages a user has to browse first before she gets into his/her own Blog Page, such as news feed or profile. Next, before the user posts any blog, she should edit the content first and then confirms the blog posting usually by clicking a Submit button. Finally, the user will get a Return Page which indicates the success of posting. In contrast, let us consider a J-worm's propagation which is also shown in the top half of Figure 5. Here we assume the J-worm is just embedded in the infected Blog Page without regard to the concrete locations, e.g., post body or comment board. And then the J-worm will silently POST itself into Blog Page of the victim's friends who just browse the infected pages. Therefore, if we compare the two UITs for the content-submissions from user and J-worm, which have been presented in the bottom half of Figure 5, we can easily identify the scenario in which J-worm is propagating as there is no edit or confirm.

Similarly, based on our discussion of E-worm in the Section 2.2, we can only detect the forged confirm for a suspicious process. Hence, if we are able to collect the related UIT for E-worm's payload-submission, we can also detect its propagation effectively.

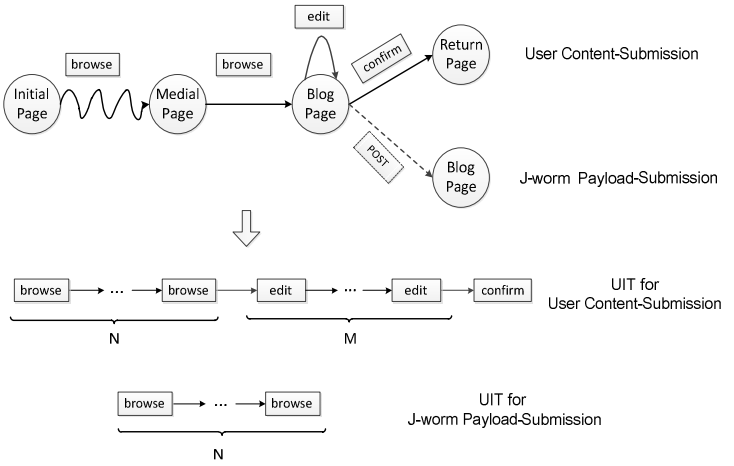


Fig. 5. User interaction traces involved in different conten-submissions

3 The OSNGuard System Architecture

In this section, we will introduce OSNGuard, a pure client-side system which aims to detect all OSN worms in real-time manner. In the following subsections, after providing an overview of the system, we elaborate the design of its components.

3.1 Overview

Figure 6 illustrates the fundamental architecture of OSNGuard system which consists of four functional modules and an extra configuration module. The Supervisor is responsible for configuring the entire system, e.g., loading all the possible locations for the content-submissions. Meanwhile, the Supervisor also manages to load or unload other modules in order to collect the relevant social traffic or user activities. The Social Traffic Monitor (STM) is in charge of sniffing network traffic to find suspicious process connecting the OSN website. Besides, the STM is also designed to trigger worm detection once the social traffic involved in a content-submission is detected. To collect the interaction traces, our Social Interaction Tracer (SIT) will capture every interaction generated from the client user when visiting OSN websites. For each collected user activity, the SIT will directly send it to our OSN Worm Detector (OWD) module which will ultimately record the activities to construct the corresponding UIT. Furthermore, the OWD will also accept normal interaction traces from the external configuration module as the user behavior model. To detect the propagation of OSN worms, our system will compare the configured traces with the collected ones.

Now we will describe the components of OSNGuard in the order of the OSN worm detection work-flow as presented in Figure 6.

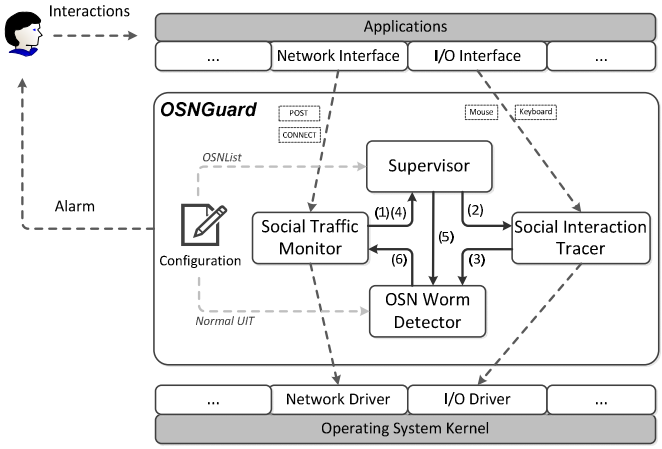


Fig. 6. Overview of OSNGuard system architecture

3.2 Social Traffic Monitor

To successfully detect OSN worms, we will first find suspicious processes by which the worms will propagate their payload. And then, we also need to identify the specific social traffic involved in a content-submission to trigger the worm detector.

Find Suspicious Processes. While it is straightforward to sniff the host traffic to find the suspicious processes that are connecting to OSN websites from the client Network Interface Card, however in OSNGuard, the STM is designed as a dynamic loadable module which will be loaded into the suspicious process space in order to precisely collect the involved information, such as the process id (PID) which will be used by the SIT to trace the corresponding user interactions.

Particularly, through Windows SPI technology, we design the STM as a service provider which will timely monitor each CONNECT intention of applications and precisely collect the process information, i.e., PID. Once finding any CONNECT to OSN website, STM will immediately signal the Supervisior upon the appearance of a suspicious process.

Identify Content-Submission. To detect any submission from a client, we need to do more works. Basically, our current STM will only filter any outbound HTTP POST request from suspicious processes as a content-submission. However, adding support for HTTP GET request as a content-submission should be straightforward. Besides, considering the popularity of SSL/HTTPS, all HTTP stream data may be encrypted and we have to be able to deal with this encrypted traffic.

Based on our thorough investigation, encrypted strategies vary from one OSN website to another. Accordingly, we use a configurable number, named EL, to refer to the encrypted level of website and it will be configured in the OSNList described in the next subsection. Specifically, for OSNs such as RenRen and Weibo, we will configure their ELs to 0 as they do not adopt any encrypting methods at all. And for OSNs such as Twitter, we will set their ELs to 2 as they encrypt all HTTP content

with SSL/TLS. Finally, we will set their ELs to 1 for OSNs such as Facebook as they only encrypt user name and password for login and do not encrypt other content such as blog or comment posts. As OSN worms only embed themselves into ordinary content, we will adopt the same SPI technology to identify POST packages outbound to OSNs whose EL<2. And for OSNs such as Twitter, we will enable an extra parsing routing that is used to intercept all contents before they are encrypted. And we have implemented current parsing routines based on Detours lib to hook the relevant encrypting functions (e.g., *cryptencrypt* for Windows XP and *SslEncryptPacket* for Windows 7).

In summary, once intercepting any content-submission by STM, the Supervisor will immediately trigger OSN Worm Detector. And if any abnormal UIT is detected, the STM will discard all relevant packages as malicious payload, which will sufficiently block the propagation of OSN worms.

3.3 Supervisor

As the administrative module of OSNGuard, the Supervisor is mainly responsible for configuration and communication. Once the system starts up, the Superviors will load all the necessary information from the extra Configuration module. Furthermore, the Supervisor is also in charge of maintaining the communications between different modules, which is fundamental component in our system.

Configuration. When OSNGuard is launched, the Supervisor will sequentially load the configurable information consisting of two parts. One is the user-defined information which mainly includes OSNList. Particularly, the OSNList includes all websites to be monitored with some attributes such as encrypted level EL and the accepted HTTP request methods.

The other configurable information is the normal behavior model which will be utilized by the worm detector to compare with the collected UITs. In this paper, we will use regular-expression-based UIT, called Normal UIT, to summarize the patterns of interactions involved in content-submissions confirmed by users.

Communication. In our system, the Supervisor is also mainly in charge of maintaining internal communications among all the components, which enables it to coordinate their executions. For instance, once it receives the notification of the appearance of a suspicious process, the Supervisor initiates the tracing communications between the SIT and the OWD. Furthermore, the Supervisor will immediately trigger the worm detection when it is notified about a new content-submission.

3.4 Social Interaction Tracer

Once a suspicious process is found by STM, SIT will be notified by the Supervisor to collect all the relevant user activities. In our system, we only need to focus on several kinds of user actions as follows:

Browse or Edit: mainly includes user interactions related to the mouse or keyboard device, such as mouse-click and keystroke. For the convenience of description, in this paper, we refer to each mouse or keyboard activity involved in the suspicious

process as a user browse or edit action respectively. Hence, we can directly capture these actions through collecting operating system input messages. Specifically, in our current implementation, we use the SetWindowsHookex function to trace all the input messages such as WM_LBUTTONDOWN and WM_KEYDOWN on Windows.

Confirm: represents a confirmation to the content-submission. Different from the activities above, confirm activity cannot be identified by system input message. Although the method proposed in BLADE [12] is able to obtain confirmations on a pop-up button, it is not efficient for our web applications. Basically, some time-consuming computations have to be introduced to correlate the mouse-click positions with the areas of a download dialog.

Instead of obtaining any position of mouse-click or UI element, we propose a novel approach to fetch the user confirm activities in OSN web pages. The basic idea is to find and hook the HTML BUTTON element, and add an extra handler to notify the user click as a confirmation. Specifically, we can get from the handle of current window, by which we can traverse to find the HTML submit button element and add the extra handler which enables us to fetch all the confirm activities from client user. In fact, this hook-based method is inspired by our code analysis of the Koobface.

3.5 OSN Worm Detector

The OSN Worm Detector (OWD) is the critical module of OSNGuard which is mainly in charge of worm detection. It has two working modes that one is UIT-Tracing and the other is UIT-Comparing. Usually, the OWD changes its working mode based on the messages received from other modules. Figure 7 illustrates the details of detection work-flow.

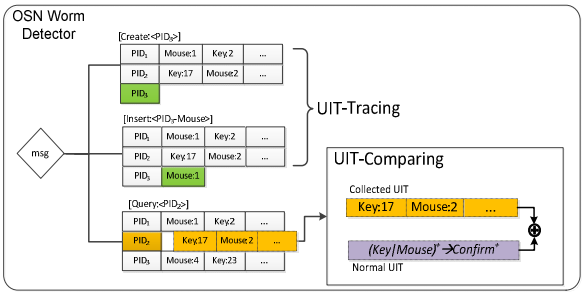


Fig. 7. The details of worm detection

UIT-Tracing. Before it receives any message, the OWD is initialized to create an empty structured table[1] which will be used to store all the UITs collected by the SIT. When receiving the first Create message from the Supervisor notified of the appearance of suspicious process, the OWD will run in UIT-Tracing mode. Usually, the PID included in the Create message will be used as the index item of a record. And then, if

[1] We use a tuple (PID, UIT) to form a complete record in a table and all the records are ordered and indexed by its PID.

a user action is found, the SIT will directly send to the OWD an Insert message which includes the PID and a collected action e.g., mouse-click, keystroke or Confirm. Once receiving this message, the OWD will immediately insert the action into the corresponding record in the table.

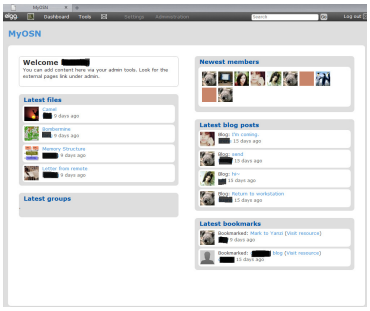
UIT-Comparing. The OWD will run in UIT-Comparing mode when it receives a Query message from the Supervisor indicating the appearance of the content-submission. In this working mode, the OWD will fetch the corresponding UIT according to the PID included in the message and compare it with the Normal UIT loaded from the configuration. Furthermore, the result will be set to true, indicating a confirmation from user, if the collected UIT matches the Normal UIT. Otherwise, the submission will be discarded as a payload-submission from OSN worms.

4 Experimental Evaluation

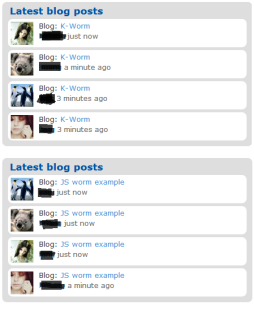
In this section, we first introduce our evaluation environment, an experimental OSN website based on an open source framework. And then, we show the effectiveness of our OSNGuard system against OSN worms. Finally, we provide test results for the performance overhead introduced by OSNGuard.

4.1 Expeirmental Environment

Considering the security threats caused by the real OSN worms, it is inconvenient to directly conduct the evaluations in a popular OSN websites. Accordingly, we build a real but small-scale OSN website, MyOSN, which is based on Elgg and deployed on a Windows machine with WampServer installed. Figure 8(a) reveals the MyOSN's news feed with various contents, e.g., newest members, latest upload files and blog posts. Furthermore, two kinds of OSN worms are also planted into several initial users' pages, e.g., blog, comment and profile. Figure 8(b) illustrates the results when worms propagate among the users without the deployment of OSNGuard. To collect real-world interaction data, more than 50 users are involved in the whole process of our evaluations.



(a) MyOSN website



(b) Propagations of OSN worms

Fig. 8. MyOSN website and two worms

4.2 System Effectiveness

In this section, we will evaluate the effectiveness of our system. Experiment 1 reveals the real-time defense against the typical Samy worm. Experiment 2 offers strong evidence that our system is also able to effectively block the propagation of E-worms such as the Koobface.

Experiment 1. OSNGuard against Samy

To evaluate the effectiveness of OSNGuard against J-worms, we plant into MyOSN the Samy worm that is provided by its author on his blog site [13]. We only modify a minimum of codes to make it infect users in various locations in MyOSN, such as the profile, blog post and comment board. From the bottom of Figure 8(b), we can see the results when the worm propagates among user blog posts without the deployment of OSNGuard.

To detect and contain this worm, we only need to configure the OSNGuard with a very simple but efficient Normal UIT:

$$(\text{mouse-click} \mid \text{keystroke})^{+} \rightarrow \text{confirm}^{+}$$

which means that a legitimate submission should be resulted from one or more confirm activities following one or more mouse-click or keystroke activities.

Table 1 provides the results of the worm detections. Due to space limit, we only presented several representative interaction traces of five users who had visited the infected pages and received the worm alerts from OSNGuard. As shown in the detected results for the worm propagation, only several mouse and/or key activities were traced when we identified the content-submissions which were actually resulted from the POST activities of the worm.

Table 1. Results of Samy detection based on user interaction traces (M=Mouse-click, K=Keystroke, C=Confirm, superscript denotes the number of repeats)

User	UIT	Result	Description
10001	K^3- M^1-K^{24}- M^1-K^{14}- M^1- C^1	√	login
	M^1-K^2- M^1-K^2-K^{15}- M^1-C^1	√	profile
	M^1-K^6- M^1-K^6-M^2-K^{78}- M^1-C^1	√	blog
	M^{13}-K^{32}-M^1- C^1	√	comment
	M^1	×	**infection**
10013	M^1-K^{14}- M^1-K^{18}- M^1- C^1	√	blog
	M^7-K^{30}- M^1- C^1	√	comment
	M^3- C^1	√	profile
	M^6-K^{53}-M^2- C^1	√	search
	M^1-K^7-M^5	×	**infection**
10025	K^3- M^1-K^{42}- M^1- C^1	√	login
	M^1-K^3-M^6-K^{24}- M^1-K^{100}- M^1-C^1	√	blog
	M^4-K^9- M^1-K^{16}-M^3- C^1	√	comment
	K^3- M^1	×	**infection**

User	UIT	Result	Description
10037	K^{28}- M^1-C^1	√	login
	K^2-M^4-K^{126}-M^1- C^1	√	comment
	K^4-M^3-K^{12}- M^1- C^1	√	upload
	M^4-K^{164}- M^1- C^1	√	comment
	M^4-K^{132}- M^1-C^1	√	comment
	M^4-K^{48}-M^1-C^1	√	search
	M^3-K^{16}-M^1	×	**infection**
10045	M^2- C^1	√	login
	M^6-K^{53}-M^2- C^1	√	comment
	M^1-K^7-M^5-C^1	√	upload
	K^3- M^1-K^{42}- M^1- C^1	√	search
	M^4-K^{12}-M^1-K^{12}- C^1	√	upload
	M^1-K^3- K^{12}- M^1- C^1	√	comment
	K^3-M^1	×	**infection**

Experiment 2. OSNGuard against Koobface

As illustrated in Figure 3(b), to launch the worm propagation, the Koobface only forges confirmation happened without any other user interaction activities, such as mouse or keyboard inputs. Accordingly, OSNGuard can also effectively identify the worm with the same Normal UIT used in the Experiment 1.

However, while it can detect the propagation of the Koobface worm, we admit that our current OSNGuard is not able to locate the stored path of the original file on client host. We need to consider the suspicious process OSNGuard can detect. In fact, it is the IWebBrowser2 COM component that will run in a standalone process and visit the OSN website, which means OSNGuard can only detect this component process. Moreover, as we investigate it with other tools such as Process Explorer, the COM component process is started up by svchost.exe process which implies that we cannot find out the original main process according to the COM component process. This finding may also explain why the attackers would like to choose the COM component as an infection vector which can be used to hide their main process.

4.3 Performance Overhead

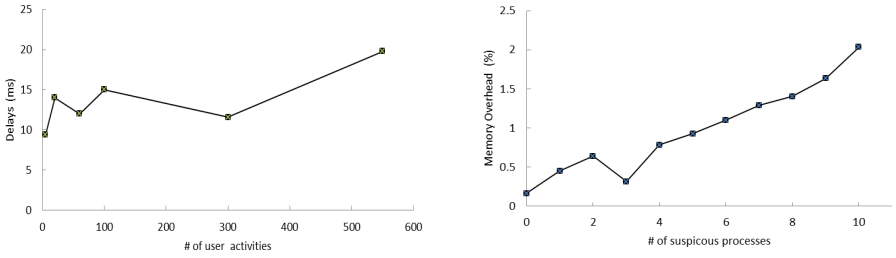
We conduct two additional experiments to measure the delay and memory consumption of OSNGuard. And both of the experiments were conducted on a Window XP client with two 2.5GHz Xeon processors and 2GB of memory.

Experiment 3. Delays on POST

Figure 10(a) shows the delays of submitting content when processing different numbers of user interaction activities. The delay for detecting a UIT containing up to 550 activities is no more than 20ms (19.8ms). Based on our practical observations, there are little users whose activity number exceeds 500 before an content-submission, which means the delay can be negligible even in a worst case scenario.

Experiment 4. Memory overhead of OSNGuard

The memory overhead introduced by OSNGuard mainly depends on the number of suspicious processes running on client system. When running without any suspicious process, OSNGuard only introduce 0.16% memory overhead, as shown in Figure 10(b). Although the overhead increases near-linearly when monitoring more suspicious processes, OSNGuard introduces less than 2.5% (2.035%) memory overhead in total when simultaneously detecting 10 suspicious processes.



(a) Delays on POST (b) Memory overhead

Fig. 9. POST delay and memory overhead introduced by OSNGuard

5 Discussion

Limitations. While the OSNGuard is designed to defend current OSN worms, we have not yet intended to prevent all current socwares [14], such as the reflected XSS and the Clickjacking. For the reflected XSS [15] attack, it usually tricks users to click on a malicious link in an e-mail message, which implies the relevant browse and confirm activities, so our OSNGuard can be ineffective in this attack scenario. For the Clickjacking [16] attack, the adversary will entice users to click a hidden button on a web page in which our OSNGuard will be also insufficient.

Future Worms. Although current OSN worms have not mainly focused on forging user interactions, we cannot ensure the effectiveness of current system when our scheme is published and especially the adversaries begin to adopt the relevant mimicking techniques. We assume that the capable attackers can forge a number of UITs. However, the personal specific behavioral biometric still cannot be forged, such as the keystroke dynamics and mouse dynamics. Hence, we argue the effectiveness of OSNGuard integrated with biometric-based authentication [17, 18] when coping with future OSN worms.

6 Related Work

OSN Worm Detection. There have been several systems proposed to deal with the J-worms in OSNs. Livshits et al. [8] provided the first automatic server-side solution, Spectator, to defend the XSS-based J-worms. They find the propagation path of the worms by tagging any HTTP request and response. If the length of the tag-path exceeded the threshold, the system would send alarms to the administrators. Sun et al. [9] proposed the first pure client side defending system which is depending on the content comparison implemented as a plugin on Firefox. By introducing view separation and request authentication, Cao et al. [7] proposed PathCutter approach which can effectively sever the paths of the propagation of J-worms.

Xu et al. [10] provided a satellite decoy network which can collect malicious messages spreading among the whole network. When the frequency of monitored messages exceeded a preset threshold, the system would notify the administrators of the propagation of malicious information. However, as we described in Section 1, their server-side system cannot protect users in real-time manner.

Drive-by Download Detection. There have been several works aiming at the drive-by download attacks, which are conceptually close to ours. Lu et al. [12] have developed BLADE, which is a system kernel extension designed to eliminate Drive-by malware installations. It asserts that all executable files delivered through browser downloads must result from explicit user consent and transparently redirects every unconsented browser download into a nonexecutable secure zone on disk.

Xu et al. [19] also provided a similar behavior based detection system, DeWare, for detecting the onset of infection delivered through vulnerable applications. It enforces the dependencies between user actions and system events, such as file-system access and process execution. Although these schemes are effective to detect drive-by downloads, they are not able to detect any Social-Engineering-based E-worms as the

worms are downloaded and installed by users themselves. In fact, the authors have admitted this limitation in their papers.

User Interactoins in OSNs. The design of OSNGuard is extremely inspired by those researches on user interactions in OSNs. Wilson et al. [20] proposed the use of interaction graphs to impart meaning to online social links by quantifying user interactions. Benevenuto et al. [21] analyzed the collected clickstream dataset to characterize user behavior in OSNs. Their analysis reveals key features such as the types and sequences of activities that users conduct on these sites. Jiang et al. [22] focused on the latent interactions such as profile browsing that cannot be observed by traditional measurement techniques.

7 Conclusions

In this paper, we have presented a UIT-based approach to prevent the propagation of OSN worms in real-time manner. To achieve this challenging goal, we divide OSN worms into two categories and summary their commonalities. It is our primary findings that OSN worms propagate themselves by accomplishing the payload-submissions with forged or without the confirmations from users. Instead of focusing on these worms, we trace all the relevant user activities and compare these to the normal behavior model to prevent malicious propagation. We have designed a client-side defending system and implemented it upon Windows platform to prove the effectiveness of our approach. Finally, we have conducted various experimental evaluations on our own OSN website and the results suggest that our client-side system can be deployed to detect and prevent OSN worms effectively.

Acknowledgements. This work was supported by the National Program on Key Basic Research Project (2012CB315804), the Major Research Plan of the National Nature Science Foundation of China (91118006), the National Nature Science Foundation of China (61073179) and the Beijing Municipal Nature Science Foundation (4122086).

References

1. Samy, `http://en.wikipedia.org/wiki/Samy_(computer_worm)`
2. Cross-site scripting worms and viruses, `https://www.whitehatsec.com/resource/whitepapers/XSS_cross_site_scripting.html`
3. Schechter, S.E., Jung, J., Berger, A.W.: Fast Detection of Scanning Worm Infections. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 59–81. Springer, Heidelberg (2004)
4. Weaver, N., Staniford, S., Paxson, V.: Very Fast Containment of Scanning Worms. In: Proceedings of 13th USENIX Security Symposium, pp. 29–44 (2004)
5. Wang, K., Cretu, G.F., Stolfo, S.J.: Anomalous Payload-Based Worm Detection and Signature Generation. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 227–246. Springer, Heidelberg (2006)

6. Ellis, D.R., Aiken, J.G., Attwood, K.S., Tenaglia, S.D.: A Behavioral Approach to Worm Detection. In: Proceedings of the 2nd ACM workshop on Rapid Malcode (WORM), pp. 43–53 (2004)
7. Cao, Y., Yegneswaran, V., Porras, P., Chen, Y.: PathCutter: Severing the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks. In: Proceedings of the 19th Network and Distributed System Security Symposium, NDSS (2012)
8. Livshits, B., Cui, W.: Spectator: Detection and Containment of JavaScript Worms. In: Proceedings of the USENIX Annual Technical Conference, pp. 335–348 (2008)
9. Sun, F., Xu, L., Su, Z.: Client-Side Detection of XSS Worms by Monitoring Payload Propagation. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 539–554. Springer, Heidelberg (2009)
10. Xu, W., Zhang, F., Zhu, S.: Toward Worm Detection in Online Social Networks. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC), pp. 11–20 (2010)
11. Elgg, http://www.elgg.org
12. Lu, L., Yegneswaran, V., Porras, P., Lee, W.: BLADE: An Attack-Agnostic Approach for Preventing Drive-by Malware Infections. In: Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS), pp. 440–450 (2010)
13. Technical explanation of The MySpace Worm, http://namb.la/popular/tech.html
14. Rahman, M.S., Huang, T., Madhyastha, H.V., Faloutsos, M.: Efficient and Scalable Socware Detection in Online Social Networks. In: USENIX Security Symposium, pp. 663–678 (2012)
15. Cross-site scripting, http://en.wikipedia.org/wiki/Cross-site_scripting
16. Clickjacking, http://en.wikipedia.org/wiki/Clickjacking
17. Monrose, F., Rubin, A.D.: Keystroke Dynamics as A Biometric for Authentication. Future Generation Computer Systems 16, 351–359 (2000)
18. Jorgensen, Z., Yu, T.: On Mouse Dynamics as A behavioral Biometric for Authentication. In: Proceedings of the 6the ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 476–482 (2011)
19. Xu, K., Yao, D., Ma, Q., Crowell, A.: Detecting Infection Onset with Behavior-based Policies. In: Proceedings of the 5th International Conference on Network and System Security (NSS), pp. 57–64 (2011)
20. Wilson, C., Boe, B., Sala, A., Puttaswamy, K.P.N., Zhao, B.Y.: User Interactions in Social Networks and Their Implications. In: Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys), pp. 205–218 (2009)
21. Benevenuto, F., Rodrigues, T., Cha, M., Almeida, V.: Characterizing User Behavior in Online Social Networks. In: Proceedings of the 9th Internet Measurement Conference (IMC), pp. 49–62 (2009)
22. Jiang, J., Wilson, C., Wang, X., Huang, P., Sha, W., Dai, Y., Zhao, B.Y.: Understanding Latent Interactions in Online Social Networks. In: Proceedings of the 10th Internet Measurement Conference (IMC), pp. 369–382 (2010)