Web Canary: A Virtualized Web Browser to Support Large-Scale Silent Collaboration in Detecting Malicious Web Sites*

Jiang Wang, Anup Ghosh, and Yih Huang

Center for Secure Information Systems,
George Mason University,
Fairfax VA 22030, USA
{jwanga,aghosh1}@gmu.edu, huangyih@cs.gmu.edu

Abstract. Malicious Web content poses a serious threat to the Internet, organizations and users. Current approaches to detecting malicious Web content employ high-powered honey clients to scan the Web for potentially malicious pages. These approaches, while effective at detecting malicious content, have the drawbacks of being few and far between, presenting a single snapshot in time of very dynamic phenomena, and having artificial test data. To address these problems, we developed a virtualized Web browser that uses large-scale collaboration to identify URLs that host malicious content on a continuing basis by building in an elective reporting system. The system, which we call a Web canary, runs a standard Web browser in a known, pristine OS every time the browser starts. Users not only report malicious URLs but also benefit from protection against malicious content. Experimental results show that it can detect the malicious Web pages effectively with acceptable overhead.

Keywords: Web browser security, honey client, malicious code, spyware, botnets, virtualization.

1 Introduction

Malicious Web content is a common vector for stealthy malicious software to infect and compromise a user's computer without his or her knowledge. In an organization, if one user's computer is compromised, the attacker can use that machine as a stepping stone to attack other computers within the organization. Some websites will deliberately host malicious content to attack users' machines in order to hijack their machines to serve as bots in a botnet, spam relays, or for fraud and other computer crime purposes. On the other hand, a website may unintentionally host malicious content, via third-party arrangements, such as Google AdSense software, user-contributed content, such as blog sites, and

* This work was supported in part by DARPA under contract W31P4Q-07-C-0244 and the National Science Foundation under grant CNS-0716323.

E. Bertino and J.B.D. Joshi (Eds.): CollaborateCom 2008, LNICST 10, pp. 24–33, 2009.

third-party software, such as visitor counters [10]. In addition, a compromised website may also host malicious code. Hosting third party content, especially advertising, is very common on the Web today, and makes finding, tracking, and black-listing websites challenging.

To actively detect these malicious websites, researchers have devised honey clients [1]. Unlike a honeypot that sits passively waiting for attacks, a honey client is usually a well-provisioned computer (or computers) driven by a program to crawl the Internet and detect malicious Web servers. It has high accuracy and can detect zero-day attacks. On the other hand, current honey clients also suffer from: 1) inadequate data about which URLs to visit; 2) temporal dynamics – they capture a snapshot in time only for the sites they visit; 3) sites that use CAPTCHAs or passwords to foil automated Web crawling; 4) following links embedded in multi-media content (such as Adobe flash or Microsoft Silverlight); 5) scalability – it is hard to crawl the entire Internet within a short time interval.

One drawback of the current honey clients is that they are driven programmatically, and cannot mimic some human behavior. In addition, honey clients tend to be few and far between because most of them are only deployed in some research laboratories. To address these issues, we employ the power of silent collaboration through Web browsing: we give the end user a honey client disguised as a Web browser. The user uses it as his or her normal browser, while our software determines if the Web browser downloaded any malicious content to the virtual machine (VM) it transparently runs in. If malicious content is detected, we notify the user, report the website to a server on the Internet, then revert the VM to its pristine state.

We take advantage of virtualization to run a snapshot of the pristine operating system (OS) every time the browser is launched. The VM has only one active application – the Web browser. So it is easy to apply anomaly detection. Perhaps more significantly, by distributing it as a Web browser, we have the ability to form a very large distributed honey client network with millions of users. While the user participates in this community of distributed honey clients, the user is also protected from the malicious content infecting his or her own OS.

We call this kind of personal honey client a 'Web canary' after the birds used in coal mines to detect poisonous gases. In our case, when the Web canary dies, or more appropriately notifies the user that an unauthorized change has been made to the VM, we capture and report the offending URL and revert the VM back to its pristine state with user approval. The Web canary browser behaves identically to a normal Web browser, except that it runs in its own VM transparently to the user.

Since the VM is restored from a snapshot when the browser is launched, it is feasible to detect malicious changes to the OS. We pre-define the normal system interaction activities of the Web browser, such as caching and cookies. The Web canary browser signals when anything else on the system changes after the browser visited a URL. Sometimes, a user may open multiple browser windows at once. In order to determine which Web page is malicious among those, we perform time correlation heuristics to correlate the unauthorized changes to the

most closely time-correlated URL. In addition, we collect these malicious URLs and send them, with user permission, to a server to record all the malicious URLs visited by Web canaries. This information, in turn, can be used by organizations, such as the Anti-Phishing Working Group, or OpenDNS to label URLs as malicious.

To demonstrate the feasibility of the Web canary idea, we built a prototype Web canary using Microsoft®Internet Explorer (IE) running on Windows XP, and VMware®Workstation. In Section 2, we describe related work and how our approach leverages prior work as well as the contribution of our work. Section 3 describes the design of Web canaries, and Section 4 describes its implementation. We evaluate the performance of Web canaries in Section 5. Section 6 concludes the paper with discussion of results.

2 Related Work

A honeypot is a trap set to attract, detect, deflect, capture, or in some manner counteract attempts at unauthorized use of information systems. Most early honeypots are server honeypots that run vulnerable software to lure attackers into attacking those servers. Recently, honey clients were developed by researchers to check Web servers for client-side exploits, such as HoneyMonkey [1], MITRE's HoneyClient [2],Capture – HPC [3] and [10].

All of above honey clients suffer from the problem that it is difficult to mimic human behavior to get past CAPTCHAs and authentication mechanisms. In addition, honey clients tend to be few and far between and only capture a snapshot of the Web in time. Given the fast-changing, dynamic nature of not only the Web, but also threats, a solution is needed that can be widely distributed while continuously capturing malicious activity from websites people are actually visiting. Web canaries enable this solution by letting the end user actually be the driver.

Since the Web canary is implemented as a Web browser, it has the potential to be widely distributed on an Internet scale. Also, Web canaries get around CAPTCHAs and authentication issues that honey clients normally suffer. Finally, the URLs visited by Web canaries represent ground truth on meaningful URLs since they represent the actual websites users visit.

Related work on sandboxing browsers can be divided into two classes: weak isolation, such as IE 7 protected mode [14], ZoneAlarm®Force Field [13], Google Chrome[20], and strong isolation, such as browser virtual appliance [17] and browser operating system [11]. Web canary provides strong isolation because of the whole system virtualization approach and also provides a detection mechanism to detect and report malicious URLs.

In addition, some researchers work in programming language analysis and browser-level add-ons to enhance the browser security, such as [6] [7] [8] [15] [18] [16]. They are complementary to the Web canary since we mainly focus on the OS level. Also, King et.al [21] are attempting to build a secure browser from scratch and SpyProxy [12] uses execution-based analysis to protect users.

Another related class of work focuses on how to use the log files from firewall collaboratively to generate black lists of potentially malicious sites based on their firewall intrusion attempts [19]. We think collaboration on intrusion records enables greater understanding of adversarial methods. However, the work in [19] focuses on firewall records, which will record blind attempts to access open ports behind firewalls, while our approach will share records about actual intrusions against a VM on end hosts.

3 Design

The goal of the Web canary is to detect and collect the malicious URLs without any specific knowledge of attacks themselves. To this end, we leverage virtualization technologies to run the common and vulnerable Web browser in a VM. Although a VM is equipped with a complete (guest) OS, its use is restricted to the browser application. Any other applications launched in the VM will be flagged as potentially malicious activity. The VM starts from a suspended state in pristine condition when the user launches the browser. This method provides fast start-up time as well as ensures the VM starts from a known good state. As the browser runs, we monitor system events in the VM. We also set up a specific folder for files downloaded by the user. Any other changes to the guest OS are treated as intrusion and we need to record the offending URL and report it to a central server.

3.1 System Monitoring and Filtering Module

Similar to other honey client solutions, a key requirement for the Web canary is a reliable and comprehensive monitor for file system and process creation events to detect potentially malicious behavior. We prefer a kernel-level monitor, because sufficient context is available in the kernel for detecting malicious websites and it is more difficult to subvert. Also, the monitor module should be able to filter out the normal behaviors of the Web browser.

3.2 URL Collecting Module

Once an unauthorized event is detected, we attempt to associate it with a particular website or URL. For this purpose, we create a browser add-on to capture the URLs requests made by the browser. Upon receiving system change events and visited URLs, we correlate them together. Correlating URLs to system changes is straightforward when Web pages are loaded one after another singly. In the presence of concurrent sessions, however, we cannot absolutely determine which one(s) of them cause malicious events. Instead, we list all the concurrently opening URLs as potentially malicious. A user can subsequently visit the listed URLs one by one in the Web canary (or use a pure honey client) and check the results. Note that there is a subtle difference between the state of multiple Web pages opened and the action of opening multiple pages at the same time. The former happens more frequently while the latter happens less frequently. Only the latter will incur some difficulties for correlating URLs to the events.

3.3 Auto-Reversion

When a malicious website is detected, the Web canary notifies the user, reports the offending website to our server on the Internet, and asks permission to revert the VM to its pristine state. We also provide an option to schedule an auto-reversion of the VM to its pristine state upon the detection of unauthorized events.

3.4 Persistent Storage

When reverting the VM after malware is detected, no information can be kept for that VM. This may cause problems if the user wants to save certain data. To address this problem, we introduce persistent storage (PS), a shared directory accessible by the VM that resides on the host file system or a networked file system. We restrict the portion of the host system visible to the guest OS only to the shared directory. This limits the access of a malicious or compromised guest OS to only the shared directory.

This approach ensures users can save documents from the temporary guest OS on to the host OS while not exposing the rest of the host system to the untrusted guest. Likewise, we use this shared directory to store our log files of monitoring module and URL collecting module running in the guest OS. In addition, we save personal browser data such as bookmarks, history, cookies by copying it to the PS before reversion and restore it after reversion.

3.5 Attacks That Web Canaries Cannot Detect

Since the web canary detects the malicious websites by detecting the abnormal changes in the guest OS, it cannot detect malicious activities that do not change system state, e.g. phishing attacks. One limitation of the current implementation is that we do not report in-memory changes to the browser process itself that stay resident (*i.e.*,changes that do not crash the browser or spawn new processes). We note that these exploits are removed after the browser is terminated or its VM reverted. In addition, our current approach may mistakenly attribute a URL as malicious when there is a time bomb and the user opened multiple Web pages.

4 Implementation

In this section, we describe how the system is implemented for Microsoft Windows XP host and Windows XP guest, using IE 6.x as the browser. Fig. 1 is the architecture for the Web canary. We use VMware Workstation^{TM}as the virtualization layer, because it is easy to install, support Windows and have live snapshot function. We also developed different versions using the free VMware server and VMware player to reduce the cost of deployment. The browser and monitor module ("Capture Client" in the Fig. 1) are running inside a VM. This implementation has two advantages: 1) the virtual machine monitor provides

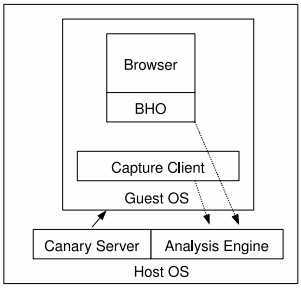

Fig. 1. Architecture of Web canary

Fig. 2. User Interface of Canary Server

strong isolation between the host OS and the guest OS. Even if the guest OS is compromised by malware, it is difficult for malware to penetrate the virtual machine monitor and infect the host OS; 2) after the Web canary detects the malware inside the guest OS, it can easily remove the malware by reverting the VM to a pristine state (snapshot). The main components for the Web canary browser shown in Fig. 1 are as follows:

1. A Brower Helper Object (BHO) which can capture all the URLs visited by IE. It then writes the URLs to a log file in PS.
2. Capture Client [3] which is able to monitor files, processes and registry changes in the Windows OS at kernel level. It also supports exclusion lists to filter out the normal changes.
3. Analysis Engine which reads the log files of the BHO and Capture Client and correlates system changes to the URL according the rule described in §3.2.
4. Canary Server which shows the health state of the guest OS (whether malware is detected or not) and can start, stop or revert the VM automatically. Fig. 2 is the GUI of the Canary Server. Canary Server can also send the detected malicious URLs to a remote central server with user's permission. The central server then saves it into a database. Currently, the central server does not authenticate the client (Canary Server) for ease of use. The central server could enforce a maximum traffic from a given user to mitigate the effect of malicious users who try to launch a DoS attack.
5. Persistent Storage (PS) which is a shared folder between the guest OS and the host OS. In the guest OS, only the BHO and Capture Client can write to this folder.

5 Evaluation

In this section, we present experimental evaluation results of the Web canary to measure its ability to detect malicious content as well as its performance overhead on a physical machine. The Web canary has not yet been released for download, so we are not able to measure its large-scale detection performance.

5.1 Testbed

Our test-bed is a Dell Precision 690 workstation with Dual-core 3.0 GHz CPU and 8GB memory. The host OS is Window XP Professional x64 edition with SP2, and the guest OS is Windows XP Professional with SP2. We use VMware Workstation 6.03. For the VM, we allocate 256 MB memory and 8 GB disk space, and installed VMware tools to support shared folder and improve user experience. The browser application is IE 6 with SP2.

5.2 Effectiveness to Detect Malicious Web Pages

To test the effectiveness of Web canaries in detecting malicious Web pages, we visited some known malicious Web pages with both the Web canary browser and the Capture Honey Client, which serves as a baseline for detection of malicious URLs. Then we compare the results.

First, we used the Capture Honey Client to find active malicious Web pages, from the potential list of malicious URLs obtained from [9] (which was scanned by Capture Honey Client one year ago). Interestingly, among all the 1937 Web pages that were previously classified as malicious by the Capture Honey Client (CHC), only 66 of them are still malicious now. This verifies our assertion that malicious websites change frequently, and using the powerful, but single honey client only provides a snapshot in time. Among the 1937 Web pages, we select all the malicious Web pages detected by CHC as well as others randomly chosen for a total of 302 Web pages. Then we use the Web canary browser and CHC to visit them one by one. The results are shown in Table 5.2.

Table 5.2 shows the number of Web pages in each class. "Malicious" is the Web pages classified as malicious by either Web canary or CHC. "Benign" means no malicious behavior detected. "Removed" means the Web page has been taken down. From Table 5.2, we can see that the Web canary browser detected all the malicious Web pages that are also detected by CHC. The numbers for benign and removed Web pages for web canary and CHC are different because CHC uses network response to attribute a Web page as "Removed" when it gets error codes 404 or 403. It cannot detect that a Web page is replaced by a removal notice page or redirect to the parent domain. However, the web canary is used by a human, so we report websites as removed when we observe this.

This experiment shows that all the malicious Web pages detected by the CHC are also detected by the Web canary. However the processes are quite different. For CHC, it first knows the URL and then visits that URL to see if anything malicious happens. For the Web canary, one of the authors inputs the URLs

Table 1. Web pages visited by the Web canary browser and Capture Honey Client

	Malicious	Benign	Removed	Total
Web canary	66	110	126	302
Capture Honey Client	66	203	33	302

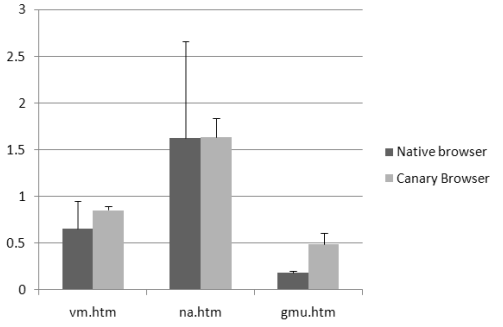

Fig. 3. Delays when loading a Web page

to visit the Web pages (one by one), then Canary Server reports detection of malicious Web pages. However, this is not the normal envisioned use of the Web canary, but rather a crafted experiment to allow us to compare detection capabilities. Normally, a user will use the Web canary as his or her regular browser. As a result, the Web canary will detect Web pages that are visited by users including those that require authentication or CAPTCHA challenges.

5.3 Performance Evaluation

Our solution is designed for personal everyday use by any user. As a result, our solution should not introduce significant delays in using a Web browser or the user will not use it. There are two types of execution performance overhead we consider: (1) the start-up time to load the VM and the browser and (2) page load time. The start-up penalty is taken every time the user starts the browser, but not for every website the user visits.

We first test page load time. The result is shown in Fig. 3. To minimize the effect of variability in network traffic between tests, we downloaded three Web pages from vmware.com, nationalgeographic.com and gmu.edu; saved it to a local Web server, and then visited each of them six times. The Fig. 3 shows the average time and the standard deviation. The black bar shows the delay for loading the Web page from a native IE 6 browser running on the host. The gray bar shows the delay for loading a Web page from the Web canary browser. The unit is sec. The Web canary browser introduces a small delay when loading vm.htm and ng.htm. The delay for loading gmu.htm from Web canary browser is bigger than that of native browser but still within 0.5 sec.

Table 2. Delays when reverting a virtual machine

	Delay for cold start (s)	Delay for warm start (s)
Web canary browser	12	7
Native browser	3.0	0.7

Table 2 shows performance results in starting the Web canary from a cold start and a warm start and similar performance measures for starting the browser natively on the host OS. The Web canary has significantly longer start-up time than the native browser (4x for cold start and 10x for warm start) because the VM software and a 256 MB memory file of the guest OS loads before the browser starts. This penalty is taken only when starting the browser for the first time, not during normal browser operation. From a user standpoint, the starting time is still on par with the times to start desktop applications, however.

6 Conclusion

In this paper, we present a system that can leverage a large, distributed network of users, who simply by using a standard Web browser, silently collaborate to detect and report malicious Web pages. In contrast to traditional honey clients, the user drives the browser as normal, but in the process, the Web canary will actively detect malicious Web content, while reporting the Web pages to a server on the Internet. The advantages of this method are that it can get meaningful URLs from users and easily visit Web pages that require CAPTCHAs or passwords. Another significant advantage is the user's machine is protected from malicious Web content that he or she may have loaded. To determine the viability of the approach, we built a prototype of Web canary. This prototype runs IE in a VMware VM, uses the client part of Capture to monitor the OS states, and a Browser Helper Object to capture URLs. We also developed an Analysis Engine to correlate the unauthorized changes to the URLs. Once a malicious Web page is detected, we provide an auto-revert feature to revert the VM to its pristine state. We also provide a persistent storage via a shared directory on the host for users to save data.

The experimental results presented here show that the Web canary can detect all the malicious Web pages detected by Capture Honey Client. It can be also used to detect the websites requiring password authentication or CAPTCHAs. In the future, we will try to correlate the unauthorized changes with particular URLs accurately even if users open multiple Web pages concurrently. We are also building a Web canary browser for Linux and other browsers such as Firefox so that it can be released and re-distributed for widespread adoption.

References

1. Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.: Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In: 13th Annual Network and Distributed System Security Symposium, Internet Society, San Diego (2006)
2. MITRE HoneyClient, `http://www.honeyclient.org/trac`
3. Capture HPC client honeypot, `https://projects.honeynet.org/capture-hpc`
4. VMware, `http://www.VMware.com`
5. Sapuntzakis, C., Lam, M.: Virtual appliances in the collective: A road to hassle-free computing. In: Workshop on Hot Topics in Operating Systems, pp. 55–60 (2003)

6. Jackson, C., Bortz, A., Boneh, D., Mitchell, J.: Protecting Browser State from Web Privacy Attacks. In: Proc. WWW (2006)
7. Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J.: Stronger Password Authentication Using Browser Extensions. In: Proc. USENIX Security (2005)
8. Zhang, Y., Egelman, S., Cranor, L.F., Hong, J.: Phinding Phish: Evaluating Anti-Phishing Tools. In: Proceedings of the 14th Annual Network & Distributed System Security Symposium (NDSS 2007), San Diego, CA (2007)
9. Know Your Enemy: Malicious Web Servers, `http://www.honeynet.org/papers/mws/`
10. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The Ghost In The Browser -Analysis of Web-based Malware. In: Proceedings of the 2007 HotBots, Usenix, Cambridge (2007)
11. Cox, R., Gribble, S., Levy, H., Hansen, J.: A safety-oriented platform for Web applications. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC (May 2006)
12. Moshchuk, et al.: SpyProxy: Execution-based Detection of Malicious Web Content - Usenix 2007 (2007)
13. ForceField (August 2008), `http://download.zonealarm.com/bin/forcefield_x/index.html`
14. IE7 Protected Mode (August 2008), `http://www.microsoft.com/windows/windows-vista/features/IE7-protected-mode.aspx`
15. Chong, S., Liu, J., Myers, A.C., Qi, X., Vikram, K., Zheng, L., Zheng, X.: Secure web applications via automatic partitioning. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007) (October 2007)
16. Howell, J., Jackson, C., Wang, H.J., Fan, X.: MashupOS: Operating system abstractions for client mashups. In: Proceedings of the Workshop on Hot Topics in Operating Systems (May 2007)
17. Browser Appliance (August 2008), `http://www.vmware.com/appliances/directory/815`
18. Reis, C., Dunagan, J., Wang, H.J., Dubrovsky, O., Esmeir, S.: BrowserShield: vulnerability-driven filtering of dynamic HTML. In: Proceedings of the 7th conference on USENIX Symposium on OSDI, Seattle, WA, November 6-8 (2006)
19. Zhang, J., Porras, P.: Highly Predictive Blacklisting. In: Proceedings of 17th USENIX Security Symposium (July 2008)
20. Barth, A., Jackson, C., Reis, C.: The Security Architecture of the Chromium Browser, Technical report (2008)
21. Grier, C., Tang, S., King, S.T.: Secure Web Browsing with the OP Web Browser. In: Proceedings of the 2008 IEEE Symposium on Security and Privacy, Oakland (May 2008)