Evaluation of a Spyware Detection System Using Thin Client Computing

Vasilis Pappas, Brian M. Bowen, and Angelos D. Keromytis

Department of Computer Science, Columbia University
{vpappas,bmbowen,angelos}@cs.columbia.edu

Abstract. Spyware – malicious software that passively collects users' information without their knowledge – is a prevalent threat. After a spyware program has collected and possibly analyzed enough data, it usually transmits such information back to its author. In this paper, we build a system to detect such malicious behaving software, based on our prior work on detecting crimeware. Our system is specifically designed to fit with thin-client computing, which is popular in some corporate environments. We provide implementation details, as well as experimental results that demonstrate the scalability and effectiveness of our system.

Keywords: Spyware, Thin Client Computing.

1 Introduction

Spyware has traditionally targeted individual consumers for purposes of conducting fraud and identity theft. Much of the defense has typically been left to anti-virus software operating on individual consumers' PCs and the financial institutions themselves who monitor for suspicious activity in an attempt to mitigate financial loss. More recently, the enterprise as has become the target [16] for spyware where the attackers' goal is to pilfer corporate information including webmail accounts, VPN accounts, and other enterprise credentials. One study conducted by RSA's FraudAction Anti-Trojan division found that almost all Fortune 500 companies have shown activity from the Zeus Trojan [12], one of the largest botnets. Given that many existing trojans and malware samples evade detection by traditional anti-virus software most of the time [12], there is demand for new approaches that can be applied at scalable levels within an enterprise.

In prior work [4], we developed a system that was designed to detect spyware proactively through the use of tamper resistant decoys. The system is intended to complement traditional signature and anomaly based defense systems rather than replace them. The system works by injecting decoys made up of monitored information that triggers alerts during exploitation. The system makes the malware's task significantly harder by requiring it to distinguish real actions from simulated actions to in order to avoid decoys. We demonstrated the system's ability to detect spyware using various types decoy credentials including those for PayPal, a large bank, and Gmail. The implementation relied upon an out-of-host software agent to drive user-like interactions in a virtual machine, seeking to convince malware residing within the guest OS that it has captured legitimate credentials. The system successfully demonstrated that decoys can be used for detecting spyware on a single host.

In this work, we explore and demonstrate the scalability of the approach across many hosts, making this work applicable to enterprise environments. Specifically, we address threats within a thin-client based environment and propose a novel architecture for bait injection on thin clients. The maturity of thin-clients has increased their usage in corporate computing environments, making this approach especially applicable [9,7]. In this system, we rely on virtualized mouse and keyboard devices to inject decoy actions and credentials to an innumerable number of hosts with very low network and CPU overhead.

In summary, the contributions for this work include:

– An extension of an already proven system that aims to proactively detect malware on a single host to one that scales to service any number of hosts.
– A thin-client based architecture that supports the injection of bait information to and from a scalable number of servers and clients.
– A demonstration of the thin-client based architecture showing that it provides reasonable performance.
– The results of experiments that examine how these new systems induce malware to exfiltrate information.

Organization: Section 2 presents previous work, related to ours. In Section 3 we describe our original system and we detail our new scalable architecture based on thin client computing. We then present our evaluation results in Section 4 and conclude in Section 5.

2 Related

The use of manually injected human input for generating network requests has been shown to be useful by Borders *et al.* [3] for detecting malware. The aim of their system is to is to thwart malware that attempts to blend in with normal user activity to avoid anomaly detection systems. Chandrasekaran *et al.* [5] expanded upon this system and demonstrated an approach to randomizing generated human input to foil potential analysis techniques that may be employed by malware. Work by Holz *et al.* [8] investigated keyloggers and dropzones, relied on executing maleware in CWSandbox [13] and automating user input with AutoIt[1] for the purpose of detecting harvesting channels. Since AutoIt resides within the host, attackers are provided with a simple means of detecting and avoiding it. In prior work, we demonstrated a platform for the automatic generation and injection of bait information designed to convince malware it has captured legitimate credentials [4]. In addition, we adapted our original system

[1] `http://www.autoitscript.com`

to personal workstation environments where the convenience of virtualization is usually absent [10]. In contrast to all prior work, this effort is focused on designing a system for the large-scale injection of decoys to detect malware that may otherwise go undetected.

Taint analysis is another technique that has been used to detect credential stealing malware [6,15]. This approach works well, but does so with a cost of a 10-20 times slowdown. Taint analysis systems also contain components that reside on the guest, which is undesirable because they can be used by malware to detect and elude the injected decoys. Our system aims to be undetectable by malware residing within so that it is not easily avoided.

The authors of [14] evaluated a number of different remote screen protocols. Although this is not directly related to the goals of our system, it is closely related to the evaluation of our system's application to thin clients.

3 Architecture

In this section, we begin by briefly presenting the goal and architecture of our original system. We then detail an architecture that demonstrates how the same approach can be scaled to handle a large number of hosts in a thin client environment, which is achieved by exploiting its centralized computation nature.

3.1 Original System

The ultimate goal of our technique is to detect crimeware using tamper resistant injection of believable decoys. In summary, we can detect the existence of credential stealing malware by (i) impersonating a user login to a sensitive site (using decoy credentials) and (ii) detecting whether this specific account was accessed by anyone else except for our system. That would be a clear evidence that the credentials were stolen and somebody tried to check the validity and/or the value of that account. Our technique depends on the following properties:

- **Out-of-host Detection.** Our system must live outside of the host to be protected. This prerequisite is for the tamper resistance feature of our system.
- **Believable Actions.** The replayed actions must be indistinguishable by any malware in the host we protect so as to not be easily eluded.
- **Injection Medium.** There must be a medium, able to transmit user like actions (mouse, keyboard, etc.) to the protected host.
- **Verification Medium.** Optionally, but highly preferable, there should be a medium that can be used to verify the injected actions. This can actually be the same medium as above, if possible.

Our original system's implementation was on a personal VM-based environment. More precisely, in order to fulfill the *Out-of-host Detection* requirement, our

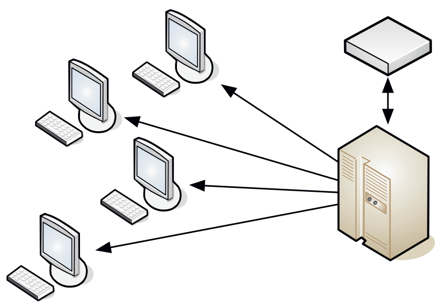


Fig. 1. Thin client environment – our system is on the top left corner

system resided on the host operating system and operated on the guest operating system(s). To verify the *Believability* of the replayed actions, we conducted a user study which concluded that the actions generated by our system were indeed indistinguishable. Moreover, as an *Injection Medium*, we utilized the X server of the host operating system to replay the actions. Finally, by slightly modifying the component of the virtual machine manager that was responsible for drawing the screen, we were able to verify the actions by checking the color value of selected pixels.

The original system relied on a language for the creation of believable actions. It is worth noting here that the approach is generic enough to be used as-is in the application bellow. This stands because the injection medium is flexible enough to support replaying of believable actions, although there could be cases where the believability of the actions can be degraded due to artifacts of the injection medium itself.

3.2 Thin Clients

The environment we chose to apply our technique to is thin clients, which, although they have been around for a long time, they are recently becoming more and more prominent in corporate networks. The main benefits of choosing such a setup are low cost, easy maintenance and energy efficiency.

A typical thin client setup consists of two main components: (i) a central virtual machine host (can be one physical server or more) and (ii) a collection of "dummy" computers connected to that host over a local and fast network. All the computation is offloaded to the central server, leaving the user terminals responsible only for transmitting user actions (keyboard, mouse, etc.) and remotely displaying the screen output of the virtual machine. Each user is then able to access and use virtual machines hosted on the central server, using these terminals (thin clients).

The application of our technique in this case was straightforward. In summary, we deployed our system like an ordinary thin client that periodically connects to each hosted virtual machine and injects decoy credentials. It is trivial to show that this type of application satisfies all the properties, previously introduced.

First, the *out-of-host* property is covered by deploying our system as a thin client and not inside the VMs under protection. Second, all the remote access protocols used in thin client environments provide a medium both for *injection* and *verification.* Figure 1 depicts what we previously described. On the lower right corner is the central server, on the left side, the thin clients and on the top right corner, our system. As our system only needs to communicate with the central server, we can safely adjust its proximity to it, reducing network overhead imposed on intermediate links.

In our prototype implementation, we assumed that there is a Linux version of the client part of the remote access protocol. For instance, in our evaluation (Section 4.1) we used VNC [11], which is a standard remote access protocol. Although this is not a requirement, it greatly improves scalability, because it allows us to easily initiate many remote access sessions, concurrently. Overall, the implementation was similar to our original system with the primary exception being that we leveraged out-of-the-box tools, as opposed to customizing. The main motivation behind that was to make our system as generic as possible and thus easily portable to other remote access protocols. More precisely, we used a vanilla version of GNU Xnee[2] for the injection of the previously recorded believable user actions, both mouse and keyboard. These actions were injected in a full screen view of the client side remote access software, Xvnc here. For the verification, we used the ImageMagick software suite[3]. More specifically, we made use of the `import` utility in order to grab arbitrary portions of the screen and the `compare` utility, to count the absolute number of different pixels. Finally, in order to enable the capability of concurrently injecting to multiple virtual machines, and thus the scalability of the system, we leveraged the Virtual Frame buffer (part of the X server). By doing this, we could simultaneously execute many full screen remote access sessions, each in a distinct X server (using the `xvfb-run` utility).

4 Evaluation

Our evaluation is divided in three parts, Subsection 4.1 examines the performance and scalability factors of our technique, when applied to a thin client environment. Next, we present the results on an exfiltration study we did using a relatively large number of malware samples. Finally, we discuss some real "hits" we had during the evaluation of our system.

4.1 Performance

In order to evaluate the performance and scalability of our system in a thin client setup, we set up such an environment in our lab. Using that as a testbed, we measured both the overhead and the limits of our system.

More precisely, we used three Dell PowerEdge R410 servers, each having 8 CPU cores, 24Gb of memory and 1 TB of storage. For the virtualization layer,

[2] Website: `http://www.gnu.org/software/xnee/`

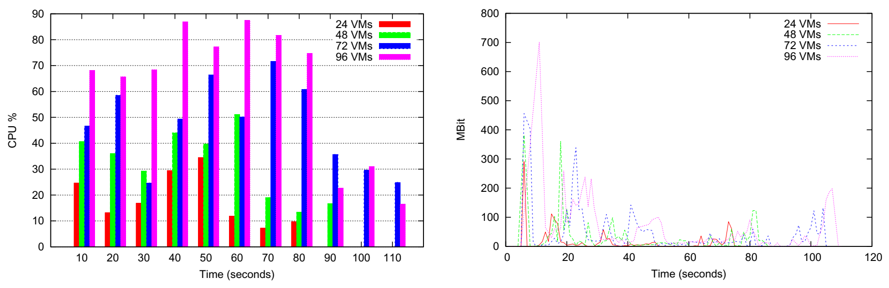
[3] Website: `http://www.imagemagick.org/`

we chose to use Xen[2,1] because it has built-in remote guest access through VNC. We installed the Xen hypervisor 4.0.0 on top of Ubuntu 10.04 server edition. On each server we hosted 32 virtual machines, running Windows XP SP2 as their guest operating system. In total, our setup was comprised of 96 virtual machines. Our prototype was also running on a virtual machine (on top of a different host), with just one CPU and 1 GB of memory.

Memory: The amount of memory required by our system is proportional to the number of concurrent sessions. Each virtual frame buffer consumes its number of pixels times the number of bytes to encode the color for each of them. For example, during our evaluation, the screen settings on the Windows guests were set to 800x600 pixels using 32-bit colors. This equals to $800 * 600 * 4 = 1,920,000$ bytes, or ∼2 MB. The total memory consumption for the whole 96 VM set is ∼176 MB.

Scalability: In the first part of our evaluation we examine the scalability of our system. In order to do that, we monitored both the network and CPU utilization, under various workloads – in terms of simultaneous injections. More precisely, the different workloads we used were 24, 48, 72 and 96 concurrent injections using our bait credentials. As for the VNC settings, we used the default values (full color and *hextile* encoding).

Figure 2(a) shows the CPU utilization under each workload. In this figure, we observe two expected things. First, the CPU load is proportional to the number of concurrently replayed sessions. Second, we notice an increase in the total duration. This increase is the result of failed verification attempts, which leads to more wait periods. These verification failures are caused both because of the virtual machine host's high load and network level congestion which causes poor refresh rates in VNC.



(a) CPU (b) Network

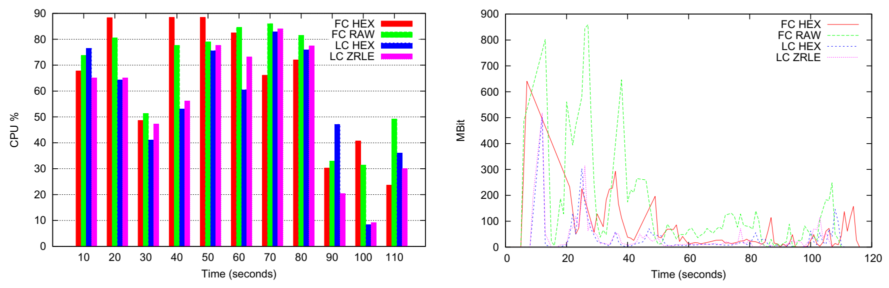
Fig. 2. CPU and network utilization when simultaneously replaying to 24, 48, 72 and 96 VMs (using full color and HEX encoding). As expected, both metrics are proportional to the number of the VMs.

The other resource we measured, in order to analyze the scalability of our system, is network utilization. Figure 2(b) shows the total network usage, under different workloads. In general, we see that network usage is high in the beginning of the injection sessions (first 30 seconds) and decreases afterwards. This is caused because VNC transmits only the portions of the screen that have been changed. In the very beginning, the whole screen has to be transmitted (first peak) and right after, Internet Explorer is started in maximized mode (high usage around the 20th second). Although the network utilization may seem forbiddingly high at times, we have to keep in mind that (i) we try to measure the scalability in the worst case scenario – that is all the injection sessions are initiated simultaneously – and (ii) this is a prototype unoptimized implementation, using of-the-self tools. The most important thing to keep from this measurement is that our system, even under these conditions, was robust enough to sustain and adapt to the workload increases.

Optimizations: After we demonstrated the scalability and adaptability to resource variations, we experimented with application level optimizations. Although we could achieve much better overall performance by developing custom injection and verification tools, we wanted to examine the benefits of tweaking parameters of the remote access protocol – VNC in this case. There are two such parameters that are related to the quality of the transmitted screen view. These are: (i) color depth and the encoding algorithm used. The different options for color depth are: 8, 256 or full colors. Each time something has changed on the screen, VNC transmits the surrounding rectangle of that portion, encoded in one of the following ways:

– `RAW`. This is the simplest out of all the encoding schemes. As its name implies, rectangles are transmitted in *width* x *height* pixel values.
– `HEXTILE`. In this case, the rectangles to be transmitted are firstly partitioned in 16x16 tiles. Then, each of them is either sent raw (as above) or using a variant of Rise-and-Run-length-Encoding, where a sequence of identical pixels are compacted to a single color value and repeat count.
– `ZRLE`. Finally, this encoding scheme combines a form of the previous one with Zlib compression.

In order to measure the benefits and tradeoffs of the different encodings and color depths, we evaluated four typical combinations. These were full color-`RAW`, full color-`HEXTILE`, 8 colors-`HEXTILE` and 8 color-`ZRLE`. For each combination, we concurrently injected bait credentials to the whole VM set – the 96 of them. As before, we collected CPU and network utilization statistics. Figure 3(a) shows the CPU usage under the different encoding-color depth pairs. Using full color yields slightly higher CPU utilization, but, overall the benefit seems negligible. On the other hand, network utilization (shown in Figure 3(b)) is indeed affected by the different encoding-color depth combinations. As expected, using full color and `RAW` encoding is the most network demanding scheme. Switching to `HEXTILE` encoding clearly results to a first improvement. Finally, lowering the color depth reduces network utilization even more. It is interesting to see that the encoding scheme does not play such a big role when using just a few colors. Hence, it would be sufficient to use even `HEXTILE` instead of `ZRLE`, in order to save a few CPU cycles.



(a) CPU (b) Network

Fig. 3. CPU and network utilization when replaying to all 96 VMs, using different combinations of color depth and encoding schemes. RAW encoding is clearly the most demanding. As for the low color depth ones, there is no big difference between HEX and ZRLE.

4.2 Exfiltration Study

In order to demonstrate the threat posed by credential stealing spyware, we conducted a study using a relatively large number of distinct samples. For the purposes of our study, we used variations of the Zeus (also known as ZBot) malware which is notorious for its credential stealing capabilities. All the samples were downloaded from Zeus Tracker[4].

In previous work, we also did provide a similar study, but somehow more limited, as each malware sample was only active on a VM for a small amount of time – order of a few tens of minutes. In our current study, we installed each malware sample on a separate VM, running on the virtualized infrastructure we built in order to simulate a thin client environment. By keeping each malware active for a relatively long period (weeks or even months) we want to explore two probable phenomena, not covered by our previous study. Firstly, we want to examine whether there are malware instances that wait for a period of time before exfiltrating the stolen credentials, and secondly, it would be interesting to see whether instances not intended to exfiltrate, get updated in a later time to do so. Both of these cases, if existent, would require a larger time window than our previous study, to happen.

We bootstrapped the study by installing all the malware samples available at the Zeus Tracker, and also we automated the procedure of installing new samples as they are made available. In total, during the study there were 108 Zeus malware instances installed on distinct VMs running on our Dell servers for a period of 3 to 4 weeks. During that time, we periodically injected both Paypal and anonymous bank's bait credentials. The component that monitors for external login attempts to the bait accounts was running for the next few months, as login attempts can occur even months after the credentials are stolen – based on our previous study. Along with the injections, we also monitored the

[4] Site: `https://zeustracker.abuse.ch`

Table 1. Top10 domain names / IP addresses that malware communicate with (left). Top10 script names that exfiltrated data are "dropped" to (right).

#	Domain / IP address	Count	Dropzone Script	Count
1	varxx.com	29808	/xt/gate.php	29808
2	nevereversite.ru	18890	/gate321.php	18890
3	95.224.124.151:555	17101	/temp/stuk.php	17820
4	65.60.36.114	13218	/∼ataactc1/z/gate.php	13218
5	podgorz.org	9599	/zuo/zsweb_cleaned/gate.php	9599
6	iesahnaepi.ru	8042	/y93/_gate.php	6238
7	wifahquaht.ru	4763	/cp11/zengate.php	4243
8	community.infinitie.net	3436	/cp01/zengate.php	2945
9	esvr3.ru	2945	/k1o/_gate.php	2892
10	phaizeipeu.ru	2702	/cache/lang_cache/web/s.php	2888

network traffic in order to see which of the malware samples have already started exfiltrating data.

Even in a such a short time period, we already encountered thousands of suspicious data transmissions. More precisely, we saw that from 74 out of the 108 VMs, outbound HTTP POST messages were transmitted to websites other than the ones we are navigating to while injecting, or even to raw IP addresses. These are most probably drop zones for the credentials stolen by the malware samples and/or configuration or command updates. In total, we recorded 134,302 such requests. The body of each POST message is in binary format, most probably encrypted in some way. Table 1 contains both the top 10 host names / IP addresses that exfiltrated data were sent to and the top 10 script names in the POST messages that handle the data, along with the number of times they appeared in our logs. By examining the counter values on both lists, we see that there are cases where there is an one to one match between host names and script names. After looking into these cases, we saw that these script names were only accessed on these host names. On the other hand, in the rest of the cases, where host name counters do not match script name counters, some scripts with the same name were installed on different hosts and some host names had more than one scripts installed.

4.3 Feasibility Study

In total, we encountered two hits on the bait accounts from the 108 installed malware samples (described in the previous subsection). The first one was on an account from the anonymous bank, after 26 days. The second hit was a Paypal account access almost two months after (57 days). These results show that our technique is indeed effective, which does validate that our new architecture is working.

As far as the number of hits is concerned, it does raise some interesting questions. On one hand, it could be normal for only a ∼2% of the accounts to be accessed. Some of the dropzones could be inactive or offline. Or, some malware samples may be unable to steal the accounts from the financial services we used, or their owners were not interested to these type of accounts, etc. On the other hand, the low hits percentage could be due to the nature of our study. One thing that we have to keep in mind is the fact that all the malware samples we used were downloaded from Zeus Tracker. As the attackers get more and more sophisticated and cautious, it would be no surprise to us that they could discard any credentials reported by malware samples that have been published in sites like Zeus Tracker. Similarly, as our main goal was the performance and scalability evaluation, the injection of the bait credentials was periodical and simultaneous to all the accounts and all the VMs were connected to the Internet through a single public IP address (NAT). It would be trivial for an attacker with several malware instances to filter out our credentials as suspicious, because they are all reported from the same IP address, periodically and simultaneously.

5 Conclusion

We presented the application of our spyware detection technique for a common setup in multiuser enterprise environments. We demonstrated it for thin client environments where we utilized out-of-the-box tools to implement our tamper resistant bait injection and action verification. The system was designed to be generic and portable to different remote access protocol stacks to make it generally applicable.

We experimentally demonstrated the scalability of our system when applied to a thin client environment. Our results showed that our system can successfully operate concurrently on a scalable number of VMs. Finally, the study we conducted using more than a hundred of malware samples revealed a number of different relationships between the malware samples and the dropzones. In addition, the relatively small number of bait account accesses from the attackers raises some interesting questions about their sophistication.

Acknowledgements. This work was supported by the NSF through Grants CNS-09-14312 and CNS-04-26623, and ONR through MURI Contract N00014-07-1-0907. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government, ONR or the NSF.

References

1. Xen website, `http://www.xen.org/`
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, New York (2003)
3. Borders, K., Zhao, X., Prakash, A.: Siren: Catching evasive malware. In: Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 78–85 (May 2006)

4. Bowen, B.M., Prabhu, P., Kemerlis, V.P., Sidiroglou, S., Keromytis, A.D., Stolfo, S.J.: BotSwindler: Tamper resistant injection of believable decoys in VM-based hosts for crimeware detection. In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 118–137. Springer, Heidelberg (2010)
5. Chandrasekaran, M., Vidyaraman, S., Upadhyaya, S.: SpyCon: Emulating User Activities to Detect Evasive Spyware. In: Proc. of the Performance, Computing, and Communications Conference (IPCCC), pp. 502–509 (May 2007)
6. Egele, M., Kruegel, C., Kirda, E., Yin, H., Song, D.: Dynamic spyware analysis. In: Proc. of the USENIX Annual Technical Conference, Santa Clara, CA, USA, pp. 233–246 (June 2007)
7. Fest, G.: Why thin is back in (March 2010), `http://www.americanbanker.com/usb_issues/120_3/why-thin-is-back-in-1014707-1.html`
8. Holz, T., Engelberth, M., Freiling, F.: Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 1–18. Springer, Heidelberg (2009)
9. Lohr, S.: Thin-client boom, finally? (July 2007), `http://bits.blogs.nytimes.com/2007/07/26/thin-client-boom-finally/`
10. Pappas, V., Bowen, B.M., Keromytis, A.D.: Crimeware swindling without virtual machines. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 196–202. Springer, Heidelberg (2011)
11. Richardson, T.: The rfb protocol, version 3.8, `http://realvnc.com/docs/rfbproto.pdf`
12. The Security Division of EMC RSA. Malware and enterprise. White paper (April 2010)
13. Willems, C., Holz, T., Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. In: Proc. of the IEEE Symposium on Security and Privacy (S&P), pp. 32–39 (March 2007)
14. Jae Yang, S., Nieh, J., Selsky, M., Tiwari, N.: The performance of remote display mechanisms for thin-client computing. In: ATEC 2002: Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference, pp. 131–146. USENIX Association, Berkeley (2002)
15. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panaroma: Capturing System-wide Information Flow for Malware Detection and Analysis. In: Proc. of the 14th ACM Conference on Computer and Communications Security, pp. 116–127 (2007)
16. Zetter, K.: Google hack attack was ultra sophisticated, new details show (January 2010), `http://www.wired.com/threatlevel/2010/01/operation-aurora/`