Crimeware Swindling without Virtual Machines*

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

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

Abstract. In previous work, we introduced a bait-injection system designed to delude and detect crimeware by forcing it to reveal itself during the exploitation of captured information. Although effective as a technique, our original system was practically limited, as it was implemented in a personal VM environment. In this paper, we investigate how to extend our system by applying it to personal workstation environments. Adapting our system to such a different environment reveals a number of challenging issues, such as scalability, portability, and choice of physical communication means. We provide implementation details and we evaluate the effectiveness of our new architecture.

1 Introduction

The existence of an underground economy that trades in stolen digital credentials drives an industry that creates and distributes crimeware aimed at stealing sensitive information. Mechanisms employed by crimeware typically include web-based form grabbing, key stroke logging, the recording of screen shots, and video capture. Depending on the features and sophistication of the malicious software, it can be purchased on the blackmarket for hundreds to thousands of dollars. The ease at which one can find such malware and the relatively low cost of this software, provides easy opportunity for those looking to trade and exploit stolen identities. The difficulty of detecting such software [4] poses a problem for everyone from home users to large corporations.

We focus our effort on the use of bait information to detect crimeware that may otherwise go undetected. Our system is designed to complement traditional detection mechanisms rather than replace them. In prior work [2], we demonstrated a system that was designed for the tamper resistant injection of decoys to detect crimeware. The system worked by forcing

* This work was supported by the NSF through Grant CNS-09-14312 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.

the crimeware to reveal itself during the exploitation of monitored information. We demonstrated the system's ability to detect crimeware 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 malware, but did so only on a single host. In addition, we adapted and evaluated our system on a thin-client based environment [5], but still our system depended on virtualization.

Prior related work includes that by Borders *et al.* [1] where they used manually injected human input to generate network requests to detect malware. The aim of their system was to thwart malware that attempts to blend in with normal user activity to avoid anomaly detection systems. Chandrasekaran *et al.* [3] expanded upon this system and demonstrated an approach to randomize generated human input to foil potential analysis techniques that may be employed by malware. Unlike our system, these did not rely on automatically injected bait that is externally monitored.

In this work, we explore and demonstrate a new approach that does away with the limitation of using virtual machines. Our scheme is aimed at environments where wireless devices (such as keyboards and mouse devices) are prevalent. Although applicability is the primary goal of this effort, we note that this approach may also provide utility in proactively defending against wireless device snooping [6], a threat model that has not yet been addressed.

In summary, the contributions for this work include:

- An extension of an already proven system that proactively detects malware on a host to one that scales to serve any number of hosts.
- A wireless device-based architecture in which simulated mouse and keyboard strokes are injected wirelessly using the Bluetooth protocol.
- An approach to running simulations that verifies the success and failure of simulated actions using traffic analysis of encrypted protocols.

2 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 capable of transmitting 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.

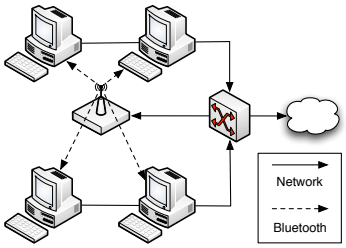

Fig. 1. Personal workstation environment – our system is in the center.

Our original system was implemented on a personal VM-based environment. More precisely, in order to fulfill the *Out-of-host Detection* requirement, our system resided on the host operating system and operated on the guest operating system. To verify the *Believability* of the replayed actions, we conducted a user study which verified that the actions generated by our system are indeed indistinguishable. Moreover, we utilized the X server of the host operating system to replay the actions – *Injection Medium*. Finally, by modifying the component of the virtual machine manager that is 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 our current work too. This stands because the injection medium was 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 Personal Workstations

In order to make the system practical in corporate computing environments, we consider how it can be applied to personal workstations. The lack of centralized control over traditional workstations poses the greatest challenge for our approach. Recall that a critical component of our system is the injection medium, which has to be deployed out-of-host in order to be tamper resistant. The approach we propose relies on simulated Bluetooth input devices as an out-of-host injection medium. In selecting Bluetooth, we leverage the physical proximity of the personal workstations to one another within a typical workspace.

One challenge for this approach is the absence of a way to monitor the screen of the hosts under protection, which was the verification medium in our original system implementation (VM-based). A naive solution to this problem would be to mirror the output of each screen using dedicated cables and monitor each of them through a video switch. However, this would be cumbersome to deploy and not scalable. The solution we adopted in this case was to move from visual verification to *Network Level Verification*. Instead of trying to verify an action by checking whether the screen output changed, we verify that some network level effects are evident. In order to apply this type of verification, we only need to mirror the network traffic and feed it to our system. This is easily applicable in most corporate environments (most enterprise level switches and routers provide this functionality out of the box) and scalable, as no physical level modifications are necessary when adding or removing workstations.

One apparent limitation of network level verification is that it can only be applied to actions that have network level effects. For instance, we would not be able to verify opening a text editor, writing some sensitive data and storing that file. However, since we are only interested in detecting crimeware that steal credentials from sensitive websites, being able to verify that a login using decoy data was successful is sufficient.

The main challenge of network level verification is that, most of the time, communication with these sensitive websites is encrypted. If, for instance, the communication was cleartext, the verification would be as trivial as searching for some specific keywords within the resulting webpage. To overcome that, we perform the verification in the following two steps: *(i)* verify that a connection to an IP address of the sensitive website's web server is established and *(ii)* watch for a specific conversation pattern, in terms of bytes sent and received. Although the first step is trivial, the second one poses a challenge.

For the implementation of our system, we utilized GNU Xnee to replay the believable actions in conjunction with a Bluetooth proxy to transmit them. The Bluetooth proxy is a modified version of Bluemaemo[1], which is a remote controller program for mobile devices that runs the Maemo OS.

Network level verification was implemented as an application on top of `libnids`[2]. It was designed to monitor the traffic and report conversation summaries. Both the Bluetooth and the network level verification components are designed to execute in a synchronized manner. The Bluetooth proxy program receives actions from Xnee, translates them to Bluetooth HID protocol, and transmits them to the host. If the action is network-level verifiable, it starts the network monitor and verifies that its output is the expected.

4 Evaluation

In this section, we evaluate the effectiveness of the network level verification. In order to do that, we collected network traffic dumps during both successful and failed login attempts. Using this data, we show that there are some network level characteristics that can be used in order to verify whether a login attempt was successful.

After manually analyzing some of the network traffic dumps, we concluded that an effective and lightweight way to achieve our goal is by comparing three metrics: the number of conversations, the number of exchanged request/response messages and the number of bytes transferred in each message. By conversations, we are referring to data exchanged between our host and the web servers of the sensitive site, excluding any other data exchanged with third party CDNs, *etc.* For example, consider the following sequence:

```
192.168.0.1 192.168.0.42  >70  <2728  >204  <67  >762  <1260
```

The first two fields represent the IP addresses of the participators and each of the next fields shows the aggregated number of bytes transmitted in each direction. Initially, `192.168.0.1` sent 70 bytes to `192.168.0.42`, who then replied with 2728 bytes and so forth.

After computing the conversation summaries for each of the successful logins to both Paypal and an anonymous bank website, we saw that the number of distinct conversations and request/response pairs was constant

[1] Website: `http://wiki.maemo.org/Bluemaemo`
[2] Website: `http://libnids.sourceforge.net/`

across different login sessions in most cases. More precisely, each login session to the anonymous bank website was comprised of only one conversation with ten messages – or, five request/response pairs. Similarly, successfully logging into Paypal resulted in several conversations, but there was always one comprised of eight messages, or, four request/response pairs. On the other hand, failed login attempts to each of these two websites resulted in clearly different conversations, in terms of number of streams, number of messages and number of bytes transmitted in each message. In more detail, an unsuccessful login attempt on Paypal produced more streams and none of them were comprised of four request/response pairs. In a similar fashion, failed login attempts to the anonymous bank website produced a number of streams, out of which, none was comprised of ten messages. Also, recall that successfully logging into the same site yielded just one stream, so the distinction was even simpler in that case.

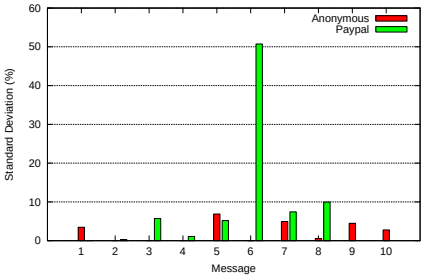

Fig. 2. Standard deviation of the number of bytes transmitted during a successful login attempt, expressed as percentage of the average – 100 attempts.

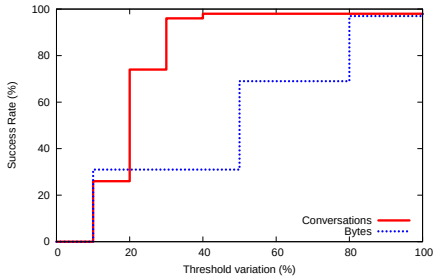

Fig. 3. Network verification success rate as a function of: (i) number of conversations and (ii) number of bytes variations.

Figure 2 shows the standard deviation of the number of bytes of each message transmitted during a successful login attempt, both for Paypal and the anonymous bank website. Odd numbered messages were sent from the client to the web server, whereas even numbered where the server's responses. In order to make it more compact and clear, the values are expressed as percentage of the average case. Although the deviation may seem large in some cases, it does not really affect our detection method because we can be permissive when choosing the thresholds. This is because the conversation matching part of the verification is performed in three levels: (i) number of conversations, (ii) number of messages exchanged and (iii) number of bytes in each message.

As a classification method, network level verification could lead to false positives and false negatives. The only case that could be labeled

as a false positive, is a failed login attempt, verified as successful by our procedure. Recall that, we did examine that simple case for both websites we used and we verified that our procedure can effectively distinguish a successful from a failed login. On the other hand, due to the dynamic nature of web and other network protocols, there could be cases where verification could fail, even though the action executed normally (false negatives). Figure 3 depicts the success rate of network level verification as a function of both the number of conversations and the number of bytes in each message – number of messages did not affect the success rate in this case. The y axis represents the percentage of successful verifications out of the 100 successful logins on Paypal. The X axis shows the percentage of variation we allowed on the average number of number of conversations (red line) and number of bytes in each message (blue line). It is worth mentioning here, that even with these "relaxed" parameters, 30% and 80%, our verification procedure had no false positives.

5 Conclusion

We presented a practical application of our crimeware detection technique on personal workstations environments. The absence of virtualization, which was an important component of the original system, made the application even more challenging. In order for our system to remain out-of-host and thus tamper-resistant, we utilized simulated wireless input devices (Bluetooth) and also developed a network level verifier for the injected actions. Finally, our experimental results on the effectiveness of the network level verification showed that it is indeed practical.

References

1. Kevin Borders, Xin Zhao, and Atul Prakash. Siren: Catching evasive malware. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 78–85, Oakland, CA, USA, May 2006.
2. Brian M. Bowen, Pratap Prabhu, Vasileios P. Kemerlis, Stelios Sidiroglou, Angelos D. Keromytis, and Salvatore J. Stolfo. Botswindler: Tamper resistant injection of believable decoys in vm-based hosts for crimeware detection. In *RAID*, 2010.
3. M. Chandrasekaran, S. Vidyaraman, and S. Upadhyaya. SpyCon: Emulating User Activities to Detect Evasive Spyware. In *Proc. of the Performance, Computing, and Communications Conference (IPCCC)*, pages 502–509, May 2007.
4. RSA (EMC's Security Division). Malware and enterprise. White paper, April 2010.
5. Vasilis Pappas, Brian M. Bowen, and Angelos D. Keromytis. Evaluation of a spyware detection system using thin client computing. In *ICISC*, 2010.
6. Thorsten Schroeder and Max Moser. Practical exploitation of modern wireless devices. `http://www.remote-exploit.org/?p=437`, March 2010.