How To Login From an Internet Café Without Worrying About Keyloggers

Cormac Herley and Dinei Florêncio
Microsoft Research, Redmond

ABSTRACT

Roaming users who use untrusted machines to access password protected accounts have few good options. An internet café machine can easily be running a keylogger. The roaming user has no reliable way of determining whether it is safe, and has no alternative to typing the password. We describe a simple trick the user can employ that is entirely effective in concealing the password. We verify its efficacy against the most popular keylogging programs.

1. INTRODUCTION

Keylogging is one of the most insidious threats to a user's personal information. Passwords, credit card numbers, PII *etc.* are potentially exposed; and the incidence of keyloggers in-the-wild is apparently growing rapidly. Unlike Phishing, this is not an attack that alert and sophisticated users can avoid. Writing a keylogger is a trivially easy task [6, 4], there are numerous freeware offerings, and many of them make efforts to conceal their presence. For example, they will not show up in the Task Manager process list. There's even a feature comparison site [1] for those interested in the hardest to detect keyloggers.

Home and enterprise users may be able to trust their systems if they maintain good firewall, anti-virus and update strategies. However roaming users have no control over what is installed. Certain internet kiosks restrict input access to the machine to prevent software installation. This makes it less likely that another user of the machine has installed a keylogger, so long as the administrator has set good policies. But this requires knowing that the administrator is both competent and trustworthy. As things stand a user has no reliable way to determine if a machine is running a keylogger or not. In this environment is there anything a user can do to protect themselves from the possibly catastrophic loss of data ?

2. A SIMPLE TRICK

We assume that the machine we use has a keylogger running. We'll also assume that it's not discoverable by the user, and that we wish to primarily protect any passwords the user types (we're less concerned about other typing). There are many ways of implementing such a keylogger, and the details won't concern us; in Windows `user32.dll` provides event handlers that any application can invoke to trap every keyboard and mouse event. There are many other approaches, and it is true for every major OS [6, 4]. Thus the keylogger gets a string that grows in length as keys are typed. For convenience, some keyloggers generate different strings for the keys that are intended for different applications. This just involves checking which window has focus at the time of the key event. It is now very easy for the keylogger to harvest passwords. The string of keys sent to the browser will often contain domain names (at an internet café most people will type domains since they are not in "favorites"), followed by userid and passwords. For example the segment

```
www.hotmail.comsarahj7@hotmail.comsnoopy2
```

tells the logger that `sarahj7@hotmail.com` has password "snoopy2" at hotmail. By parsing the string for common domains such as hotmail, paypal, amazon, fidelity, the task is made even easier.

At first our task may seem impossible: if the keylogger sees everything how can we hide the password from it? Rather than hide the password our approach is to embed it in a sequence of random characters. So we seek a way of entering random keys so that they will be seen by the keylogger, but will not affect normal login. The trick lies in the fact that keyloggers employ very low level OS calls. The keylogger sees everything, but it doesn't understand what it sees. The browser also sees everything, but it doesn't use everything that it sees: *it does not know what to do with keys that are typed anywhere other than the text entry fields, and lets them fall on the floor.* The keylogger has no easy way to determine which keys are used by the browser and which fall on the floor. It is very easy to record all of the keys or mouse events (this is true both for Windows and Linux based systems [4, 7]). It is also very easy to determine which application had focus at the time of the event (*e.g.* this key went to the browser). But it is very hard to determine what the application did with those events.

Between successive keys of the password we will enter random keys. In the spirit of chaffing and winnowing [5], the string that the keylogger receives will contain the password, but embedded in so much random junk that discovering it is infeasible. Observe that we are not exploiting a particular feature of any particular browser: this trick works with all versions of Internet Explorer, Netscape Navigator and Mozilla Firefox. We are exploiting the difficulty from the OS layer of determining how the GUI of an an application handles events. Here, then is the method:

```
Navigate to the login page desired;
Type in the userid;

for (each pwd character){
    Give focus to anywhere but the pwd field;
    Type some random characters;
    Give focus to the pwd field;
    Type the next character of the pwd}
Submit;
```

It involves typing random characters between successive characters of the password, and changing focus to and from the password field using the mouse. Instead of the password `snoopy2` the keylogger now gets:

`hotmail.comspqmlainsdgsosdgfsodgfdpuouuyhdg2`

Here a total of 26 random characters have been inserted among the 7 characters of the actual password. In general a total of n extra characters in a length k password will yield so many possible passwords that attack is infeasible (recall the password that can only be tested by attempting login). There are various attacks on this method as we explain below. However, none of the keyloggers reviewed in [1] appear to have to functionality to defeat this simple trick.

2.1 On Screen Keyboards

Rather than have users key in their passwords some web sites have experimented with on-screen keyboards as a method of secure data entry. Like our trick this forces keyloggers to do screen captures at every mouse click or every key event. One security startup [2] is offering on-screen keyboard login as a service offering to banks. Again, this relies on the fact that a non-trivial increase in the resources consumed would be required to capture these passwords. The same is not true of the on-screen keyboard offered by Windows XP Accessability tools (this is available under Programs-Accessories-Accessability Tools-On Screen Keyboard). Unfortunately this emulates keystrokes and sends them to the application that has focus. Even the simplest keylogger will catch all of the entries from the On screen keyboard as though they were typed.

3. RESULTS, LIMITATIONS, DIRECTION

We tested five shareware or commercial keylogging programs: HomeKeylogger 1.70, GhostKeylogger, KGBKeylogger, Spytector 1.2.8 and ProBot. None of them captured passwords entered using the trick we describe.

It bears pointing out that this is not a universal durable solution to the problem of keylogging. There are many tricks in the Security space that work well when used by a small number of people, but which will not withstand the attacks that a large deployment can be expected to bring. The security here comes from the fact that figuring out what an application does with keys is non-trivial for a layer of code that is below that application. Doing a screen capture *at every keystroke* will reveal which of the keys typed using this method belong to the password (the password field of the browser indicates how many keys have been typed). But we point out that taking a per-keystroke screenshot greatly increases the spyware's resource consumption (and hence it's risk of discovery) and harvesting of passwords becomes more difficult to automate.

Nonetheless, the simple mechanism of embedding the password in random keys to be extracted elsewhere is valuable. Here we inserted the random keys manually, and "extracted" them by knowing what the browser allows to fall on the floor. We pointed out that this can be attacked (though it suffices to give real protection to real users today). A truly secure approach is to have the random keys extracted somewhere other than the untrusted machine. In [3] we demonstrate how this can be done using a simple proxy server. The user again enters the password embedded in random keys, and the proxy extracts the random keys using a secret shared between the the user and proxy. In this way we can entirely avoid leaving any information about the password on the untrusted machine. Spyware that logs the keys, captures screenshots and monitors all network traffic would still be unable to discover the password without the shared secret. Details and variants are in [3].

4. REFERENCES

[1] `http://www.keylogger.org`.

[2] `http://www.bharosa.com`.

[3] D. Florêncio and C. Herley. Entering Passwords on a Spyware Infected Machine Using a Shared Secret Proxy. *MSR Tech. Report*, 2006.

[4] S. McClure, J. Scambray, and G. Kurtz. *Hacking Exposed*. McAfee, fifth edition, 2005.

[5] R. Rivest. Chaffing and Winnowing: Confidentiality without Encryption. 1998. `http://theory.lcs.mit.edu/~rivest/chaffing.txt`.

[6] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals*. Microsoft Press, 2005.

[7] E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall, 2004.