A Mechanism for Protecting Passwords from Key Loggers

Lior Shamir
Department of Computer Science
Michigan Tech
1400 Townsend Drive, Houghton, MI 49931,
{lshamir@mtu.edu}

October 11, 2005

Abstract

A simple mechanism that protects passwords from being stolen by malicious key loggers is proposed. The described mechanism is based on feeding potential key loggers with incorrect passwords by generating faked keystrokes. Applications that require their users to type in passwords (e.g. internet browsers surfing to password protected web sites) can use this mechanism so that passwords can be protected even in the absence of an anti-key logger. Different approaches of protection are proposed for thread based, hook based and kernel based key loggers.

1 Introduction

Software based key loggers are stealth surveillance applications that keep records of the user keyboard activities. By logging the user keystrokes, key loggers can capture personal information such as bank account numbers, credit card numbers, usernames, passwords, home addresses, etc, that can be used for non-legitimate purposes such as unauthorized activities in bank accounts and might provide an infrastructure for identity theft. In the popular MS-Windows operating system they commonly do not show up in the "Task Manager", and their executable files use random names and are hidden among the system files so that even expert users find it difficult to know whether a key logger is active in their system. While their potential damage is definitely a concern, simple yet effective key loggers are relatively easy to implement, and even moderately sophisticated computer programmers can fairly easily create their own malicious pieces of software.

One common anti-key loggers approach is scanning the system for known key loggers. This approach is somewhat similar to anti-virus programs. However, due to the on-going activity of key logger developers, new key loggers are introduced rapidly so that absolute protection cannot be guaranteed.

Another approach is by using anti-hook techniques and detecting processes trying to intercept messages on their way to their destination window [Aslam et al. 2004; Xu et al. 2005]. This approach has a relatively high rate of false positives [Aslam et al. 2004].

The main downside of anti-key logger programs is perhaps their dependency on the users maintenance of the system. I.e., only if the user installs an anti-key logger and reacts properly to its warnings, applications that require passwords (such as internet browsers surfing to password protected web sites) can avoid unwillingly sharing passwords with operators of malicious key loggers. If an anti-key logger is not installed, applications cannot protect their users' passwords from key logging attacks.

In this paper we propose mechanisms that allow applications to protect passwords or other valuable small pieces of information from different types of key loggers by feeding them with false data. These mechanisms can protect passwords even if an anti-key logger is not installed. In Section 2 we give a general description of the different types of key loggers, in Section 3 a protection from thread based and hook based key loggers is described, in Section 4 a protection from kernel based key loggers is proposed and in Section 5 the effectiveness of the faked password is discussed.

2 Key Loggers

Some key loggers are marketed as legitimate tools for tracking employees or family members [Blazing; Spector; Invisible; Keysnatch]. However, despite the legitimacy of some keystroke loggers, malicious keyboard monitors in the form of worms, spyware or Trojan horses are considered a concern in the field of information security [Stafford 2004]. Although key loggers are a serious threat to the security of computer users, they do not receive the same attention as viruses and spyware, despite severe criminal cases of malicious use of key loggers such as the identity theft from Kinkos customers [Network Security 2003], the password stealing from Microsoft's Redmond, WA office [Farrow 2003] and more [Baldwin & Klingdon 2003; Levine 2004].

Key loggers can be divided into three major categories: Kernel bases, thread based and hook based [Xu et al. 2005]. Kernel based key loggers are device drivers that read directly from the keyboard driver. The keystrokes are then sent to a hidden process that records the data before sending them to its evil master. Writing and installing a kernel key logger can be fairly easy in operating systems that allow direct memory access such as MS-DOS (Disk Operating System). However, in the popular MS-Windows operating system the installation of this type of key loggers requires Administrator privileges, and writing kernel drivers require a high level of programming proficiency. Therefore, key loggers of this type are extremely rare.

Unlike kernel based key loggers, thread based key loggers are much simpler to program, and even a moderately skillful computer programmer can easily write a key logger of this type. Thread based key loggers are based on a hidden thread that reads key strokes intended for other threads. In MS-Windows operating system this can be done by simply calling the *GetKeyState* and *GetKeyboardState* API functions. Users typing their password or username while using a certain application are not aware of another hidden process that monitors and records the keystrokes by constantly checking the keyboard state. An example application of a key logger that uses this approach can be found in [WinMacro].

The most common key logging approach in Windows operating system is by using a hook, which is a mechanism by which processes can intercept messages before they reach their destination. After reading the information delivered by the message, the hook lets the message continue to its final destination. By using a hook, malicious processes can read keyboard messages (e.g. WM_KEYDOWN) sent to a specific key window such as a password textbox. Hooks are familiar to software developers in the form of external programming and debugging tools such as Microsoft's Spy++ or Borland's WinSight.

3 Simple Protection from Thread Based and Hook Based Key Loggers

Passwords and usernames are highly valuable small pieces of information. While usernames are often public, passwords are kept privately and are not to be shared with others. Password theft can result in severe damage caused by the unauthorized access of the attacker. With the absence of an anti-key logger, malicious keyboard monitors secretly operating on one's system can steal passwords and transmit them to their evil masters over the internet. In order to improve their protection from key loggers, applications should protect passwords being entered by users from being stolen by key loggers, rather than simply assume that an anti-key logger is operating in the system, an assumption that is often untrue.

The basic idea of the protection is that applications simulate keystrokes by calling API functions such as *SetKeyboardState* and *keybd_event* while the user enters the password. This, of course, will result in an incorrect password entered to the password textbox. However, since the application knows which keystrokes were simulated, it can recover the original password simply by removing the characters added by the faked keystrokes. Thread based and hook based key loggers cannot tell between real keystroke and the simulated keystrokes, so they capture a wrong password that includes the real characters, but also characters added by the simulated keystrokes. The following code demonstrates a simple implementation of this approach:

```
#define MAX_FAKED_PASSWORD_LENGTH 1024
char faked_keys[MAX_FAKED_PASSWORD_LENGTH];
unsigned short password_keys_num=0;

void onKeyUp()
{  WORD random_key;
   int i,faked_keys_num;
   if (PasswordWindow->Text.Length<=password_keys_num) return; // ignore
        //  keyboard events generated by faked keystrokes
   password_keys_num++;
   faked_keys_num=1+random(10); // select a number of up to 10 faked keys
   for (i=0;i<faked_keys_num;i++)
   {  Sleep(random(250));  // a delay of a random duration
      random_key=48+random('Z'-'0'); //pick a random alphanumeric virtual key
      keybd_event(random_key,0,0,0);  // simulate key down
      Sleep(100+random(100));  // cause a delay of a random duration
      keybd_event(random_key,0,KEYEVENTF_KEYUP,0);  // simulate key up
      faked_keys[password_keys_num++]=1; // mark this key as faked
   }
}
```

The function *onKeyUp* is called when a keyboard message arrives to the window of the password textbox (*PasswordWindow*), and simulates several random keystrokes. Thread or hook based key loggers record those faked keystrokes so that the password recorded by them is different than the actual password that was entered by the user. If a fixed number N of false keystrokes are simulated when a key is pressed, a potential key logger can easily deduce the actual password by simply collecting each N*th* character in the password. In order to avoid that, the number of faked keystrokes simulated after each actual keystroke is random. The duration in which the key is being held down and the delay between the strokes are also determined randomly in order to prevent key loggers from using certain patterns of the keystroke timing [Gunetti & Picardi 2005]. The purpose of the array *faked_keys* is to record the faked keystrokes so they can later be removed when the password is submitted. This can be done using the following code:

```
void onPasswordSubmit()
{  int i;
   for (i=0;i< PasswordWindow->Text.Length;i++)
     if (!faked_keys[i])
        RealPassword=RealPassword+PasswordWindow->Text[i];
   SubmitPassword(RealPassword);
}
```

The code simply removes the characters added to the password by the faked keystrokes (each entry in *faked_keys* is a flag indicating whether the character is real or faked) before submitting the password.

For example, if the user types in the real password "abc", each actual keystroke generates a random number of faked keystrokes so that the password recorded by a malicious key logger could be something such as "aq14tbbruco9pmpo". The characters *a,b* and *c* appear in the faked password, but several random characters generated by simulated keystrokes appear between them so that the password obtained by the key logger is different than the real password. Before submitting the password, the application will know to remove the faked characters and recover the real password. The key logger, however, will lack this information and try to use the password "aq14tbbruco9pmpo", which is incorrect. The user, of course, is not aware of the simulated keystrokes and simply enters the password normally.

4 Protection from Kernel Based Key Loggers

While thread based and hook based key loggers can be tricked by the mechanism described in Section 3, kernel based key loggers reading directly from the keyboard driver inherently avoid reading faked keystrokes messages. Although kernel based key loggers are far less common than thread based and hook based keyboard monitors, their existence is still a matter of concern that should be addressed.

One approach to password protection from kernel based key loggers is by using a device driver that simulates the keystrokes by writing directly to the keyboard driver. Applications can use this driver to simulate keystrokes as described in Section 3, but instead of calling to Windows API functions such as *keybd_event*, the keystrokes can be simulated by using the device driver. Even if a kernel based key logger operating in the system is aware of this mechanism, it cannot distinguish real keystrokes from faked ones, so the effectiveness is similar to the thread based and hook based protection mechanism described in Section 3. However, the installation of such drivers requires Administrator privileges, so non-Administrator installations might not be able to use this mechanism.

Another approach that does not introduce particular installation requirements is by displaying random characters instead of simulating keystrokes, and asking the users to type those characters while typing in their password. The user will simply type the characters displayed on the screen after typing each of the real password characters. For instance, if the password is "ab", the application can ask the user to type a sequence of random characters such as "cf1" after they type the first character "a", and another sequence of random characters such as "p4qzh" after they type the second password character "b". The application knows which and how many characters the user was asked to type, so it can know to remove those characters when the password is submitted. Kernel based key loggers cannot know which of the typed characters are real password characters and which are characters that were typed as a result of an explicit request from the application. The implementing of this mechanism is very similar to the code described in Section 3, but instead of calls to the API function *keybd_event*, the simulated faked keys will simply be displayed on the screen so users can manually type them in while entering their password. The obvious downside of this approach is that users must be aware of the protection mechanism and follow the instructions by typing the faked keys while entering their password. As simple as the instructions are, home users might find this approach annoying.

5 Hacking the Faked Password

Although the proposed approach aims to prevent key loggers from stealing the actual passwords, the stolen faked passwords provide essential information that can be used by malicious attackers in order to deduce the real password. Assuming the malicious attacker is familiar with the proposed mechanism and also knows the length of the password (N) and the maximum number of faked keys that are simulated after each real keystroke (M), the attacker can know that the position of the first character is 1, the position of the second character must be greater than 1 and smaller or equal to $1 + (M + 1)$, the position of the third character is greater than 2 and smaller or equal to $1 + 2 \cdot (1 + M)$, etc'. Therefore, the maximum total number of possible combinations is $\Pi_{c=2}^{N}[1 + c \cdot (1 + M) - c] = N - 1 + (N - 1) \cdot \frac{2M+NM}{2}$. For instance, if the length of the password is 8 and the application simulates a maximum number of 10 faked keystrokes after each actual keystroke, the attacker will deduce the right password after a maximum number of 357 attempts. This is certainly a small number of attempts that can be performed even manually, without using automatic tools. However, sensitive password-protected data sources such as bank accounts use mechanisms that can detect an intensive brute force effort to hack a password, and block the account after a small number of failed attempts to log in (typically 3 failed attempts will result in automatic blocking of the account). Using the numbers of the example given above, the chances of the intruder to complete a successful log in are less than 0.01, and that number significantly decreases as the number of simulated faked keystrokes is increased.

6 Conclusion

In this paper a mechanism that protects password or other valuable small pieces of information from being stolen by key loggers was described. The main advantage of the proposed approach is that it allows applications to be protected from password theft, even if an anti-key logger is not installed in the victim computer. The mechanism does not provide absolute protection, and the chances of key logger operators to guess a stolen faked password are much higher than simply guessing the password without using the information captured by the key logger. However, this method can provide a reasonable protection in systems that block user accounts after several failed attempts to log in, and introduce an additional significant hassle to key logger operators.

References

[Aslam et al. 2004] ASLAM, M., IDREES, R.N., BAIG, M.M. AND ARSHAD, M.A. 2004. Anti-Hook Shield against the Software Key Loggers. In Proceedings of Nat. Conf. of Emerging Technologies 2004. 189–191.

[Baldwin & Klingdon 2003] BALDWIN, R.W. AND KLINGDON, K.W. 2003. Survey of Spyware and Countermeasures. Plus Five Consulting, Inc. http://www.plusfive.com/reports.html

[Blazing] Blazing Tools Software, Perfect Keylogger, http://www.blazingtools.com/bpk.html.

[Farrow 2003] FARROW, R. 2003. Is your Desktop being Wiretapped? Network Magazine 18, 8, 52–53.

[Gunetti & Picardi 2005] GUNETTI, D. AND PICARDI, C. 2005. Keystroke Analysis of Free Text. ACM Trans. Info. Syst. Security 8, 3, 312–347.

[Invisible] Invisible Keylogger, http://www.amecisco.com/iks2000.htm.

[Keysnatch] Keysnatch keylogger, http://www.fileheaven.com/Keysnatch/download/2975.htm .

[Levine 2004] LEVINE, J.R. 2004. Written Comments of Dr. John R. Levine. U.S. Senate Committee on Commerce, Science and Transportation, http://commerce.senate.gov/pdf/levine032304.pdf

[Network Security 2003] Network Security. 2003. Man steals passwords with keystroke logger. Network Security 8, 20–21.

[Spector] Spector keylogger, http://www.spector.com.

[Stafford 2004] STAFFORD, T.F. 2004. Spyware: The Ghost in the Machine. Communications of the Association of Information Systems 14, 291–306.

[WinMacro] WinMacro, http://www.phy.mtu.edu/~lshamir/downloads/WinMacro/WinMacro.html.

[Xu et al. 2005] XU, M., SALAMI, B., AND OBIMBO, C. 2005. How to Protect Personal Information against Keyloggers. Proceedings of Internet and Multimedia Systems and Applications (IMSA05), 275–280.