Bait your Hook
A Novel Detection Technique for Keyloggers

Ottawa, 15th September 2010

Stefano Ortolani - ortolani@cs.vu.nl
Cristiano Giuffrida - giuffrida@cs.vu.nl

Vrije Universiteit
Amsterdam, The Netherlands

Bruno Crispo - crispo@disi.unitn.it

Università di Trento
Trento, Italy
Keyloggers, a real threat?
Keyloggers, a real threat?

Protecting Yourself From Keylogging Thieves

TOM ZELLER Jr.
Published: February 27, 2006

The network security firm Sophos estimates that an unprotected computer has a 40 percent chance of being infected by a malicious worm within 10 minutes of being connected to the Internet. After an hour, the odds rise to 94 percent.

That's reason enough to keep up to date with operating system patches, invest in a solid antivirus program and use a basic firewall. But even with those measures in place, malicious code — including a keylogger — can sometimes find its way onto your computer.

"There are plenty of ways to get around all of those things," said Ken Dunham, director of the rapid response team at iDefense, a unit of VeriSign that focuses on computer security information.

Most major commercial antivirus software will seek out keylogging Trojan horses, as will most of the leading antispyware packages — although they may not catch them all. Some products, like Spyware Doctor from PC Tools and SpySweeper from WebRoot Software, pay particular attention to keylogging Trojans and cost about $30.

StrikeForce Technologies, based in Edison, N.J., is developing an anti-keylogging toolbar for the Internet Explorer Web browser, called WebSecure, that promises to encrypt text...
Keyloggers, a real threat?

A Strong Password Isn’t the Strongest Security

By RANDALL STROSS
Published: September 4, 2010

MAKE your password strong, with a unique jumble of letters, numbers and punctuation marks. But memorize it — never write it down. And, oh yes, change it every few months.

These instructions are supposed to protect us. But they don’t.

Some computer security experts are advancing the heretical thought that passwords might not need to be “strong,” or changed constantly. They say onerous requirements for passwords have given us a false sense of protection against potential attacks. In fact, they say, we aren’t paying enough attention to more potent threats.

Here’s one threat to keep you awake at night: Keylogging software, which is deposited on a PC by a virus, records all keystrokes — including the strongest passwords you can concoct — and then sends it surreptitiously to a remote location.

“Keeping a keylogger off your machine is about a trillion times more important than the strength of any one of your passwords,” says Cormac Herley, a principal researcher at Microsoft Research who specializes in security-related topics. He said antivirus software could detect and block many kinds of keyloggers, but “there’s no guarantee that it gets everything.”
Keyloggers, a real threat. Really?

Source: [http://www.verisign.com](http://www.verisign.com) - 2005
Keyloggers, which ones?

Unprivileged - Hook Based: 29%
Unprivileged - Cyclic Request: 5%
Privileged - Kernel Driver: 66%

Source: [http://www.securityfocus.org](http://www.securityfocus.org) - 2007
Keyloggers, a real threat! Why?

- Keyloggers are easy to develop and execute!
- They do **NOT** require any particular privilege either on installation nor during execution.
- With a managed language, e.g. **C#**, less than **100 LOC**.
Keyloggers, a real threat! Why?

```csharp
public void Intercept() {
    LowLevelKeyboardProc _proc = HookCallback;
    IntPtr _hookID = IntPtr.Zero;
    using (ProcessModule curModule = curProcess.MainModule) {
        _hookID = SetWindowsHookEx(WH_KEYBOARD_LL, proc, GetModuleHandle(curModule.ModuleName), 0);
    }
}

private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam) {
    if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN) {
        StreamWriter sw = File.AppendText(path);
        int vkCode = Marshal.ReadInt32(lParam);
        switch (((Keys)vkCode)) {
            case Keys.Space:
                sw.Write(" ");
                break;
            case Keys.Return:
                sw.WriteLine(" ");
                break;
            ...
            default:
                sw.Write(((Keys)vkCode).ToString());
        }
        sw.Close();
    }
    return CallNextHookEx(_hookID, nCode, wParam, lParam);
}
```

Easy!
Why so easy?

- Modern operating systems provide the developer with APIs to intercept keystrokes:
  - Win32 - SetWindowsHookEx(idHook, lpfn, hMod, dwThreadId)
  - X11 - gdk_window_add_filter(GdkWindow, function, data)
- The reasons:
  - Keyboards with additional, i.e. hardware defined, keys.
  - Window managers with system-defined shortcuts, e.g. Alt-Tab.
  - User applications running in the background, e.g. note-taking applications.

Advantages for a developer

**Easy to develop:** no kernel programming involved.

**Easy to deploy:** no privileges are required.

**Easy to use:** the user does not need superuser privileges.
Why so easy?

- Modern operating systems provide the developer with APIs to intercept keystrokes:
  - **Win32** - `SetWindowsHookEx(idHook, lpfn, hMod, dwThreadId)`
  - **X11** - `gdk_window_add_filter(GdkWindow, function, data)`
- The reasons:
  - Keyboards with additional, i.e. hardware defined, keys.
  - Window managers with system-defined shortcuts, e.g. Alt-Tab.
  - User applications running in the background, e.g. note-taking applications.

Advantages for a developer

**Easy to develop:** no kernel programming involved.

**Easy to deploy:** no privileges are required.

**Easy to use:** the user does not need superuser privileges.
Why so easy?

- Modern operating systems provide the developer with APIs to intercept keystrokes:
  - **Win32** - `SetWindowsHookEx(idHook, lpfn, hMod, dwThreadId)`
  - **X11** - `gdk_window_add_filter(GdkWindow, function, data)`
- The reasons:
  - Keyboards with additional, i.e. hardware defined, keys.
  - Window managers with system-defined shortcuts, e.g. Alt-Tab.
  - User applications running in the background, e.g. note-taking applications.

Advantages for a malware developer

**Easy to develop:** easy to integrate a keylogger in a malware.

**Easy to deploy:** there is no need for a vulnerability to exploit.

**Easy to use:** again, no particular privileges are required.
Why so easy?

- Modern operating systems provide the developer with APIs to intercept keystrokes:
  - **Win32** - SetWindowsHookEx(idHook, lpfn, hMod, dwThreadId)
  - **X11** - gdk_window_add_filter(GdkWindow, function, data)
- The reasons:
  - Keyboards with additional, i.e. hardware defined, keys.
  - Window managers with system defined shortcuts, e.g. Alt-Tab.
  - User applications running in the background, e.g. note-taking applications.

**Advantages for a malware developer**

**Easy to develop:** easy to integrate a keylogger in a malware.

**Easy to deploy:** there is no need for a vulnerability to exploit.

**Easy to use:** again, no particular privileges are required.
Indeed a Flaw! Countermeasures?

- **Signature-based approaches**
  - Can not cope with ever-growing 0-day keyloggers [CJ04].

- **APIs tracing and detection**
  - Requires super-user privileges and prone to false positives.

- **Taint analysis**
  - Privileged. Prone to a plethora of false positives [SB09].

- **Dynamic code instrumentation**
  - Privileged. Checking all the execution’s paths is hard [MKK07].
Indeed a Flaw! Countermeasures?

- **Signature-based approaches**
  - Can not cope with ever-growing 0-day keyloggers [CJ04].

- **APIs tracing and detection**
  - Requires super-user privileges and prone to false positives.

- **Taint analysis**
  - Privileged. Prone to a plethora of false positives [SB09].

- **Dynamic code instrumentation**
  - Privileged. Checking all the execution’s paths is hard [MKK07].
A Real Countermeasure

- Existing approaches are not enough. An ideal approach should be:
  - **Unprivileged**, hence can be run by any user on any machine.
  - **Reliable**, hence not prone to false positives.
  - **Portable**, hence easy to be coded for another OS.
- We shall pose to ourself the following question:
  - Is it possible to create the footprint of a keylogger?
Many Keyloggers, One Behavior

- A Keylogger will always log the keystrokes being issued to the system!
- Hence, we expect a correlation between:
  - The number of keystrokes the user issues.
  - The bytes the keylogger outputs by logging such keystrokes.
Our Approach

• We launch a **bait**, that is we taunt the keylogger with some input that looks real.
  • We call the process of forging the bait **Generation phase**.

• Our strategy comprises then of two contemporary phases:
  • **Injection phase** - the launch of the bait, i.e. the injection of the keystrokes.
  • **Monitor phase** - in which we monitor all the processes.

• A fourth phase, termed **Detection phase**, flags as a keylogger any process exhibiting **high** correlation between:
  • The stream of keystrokes we injected.
  • The stream of bytes the process wrote.
  • However, reasoning on raw streams is hard.

**NO superuser privileges required!**
Dealing with streams

- In order to reason over streams of data, we adopt an abstract representation termed Abstract Keystroke Pattern (AKP) form.
  - An Abstract Keystroke Pattern \( P \) is a set of samples \( P_i \).
  - Each sample \( P_i \) is the normalized amount of data measured at the time interval \( i \).
  - The normalization scales all the samples within the range \([0, 1]\).

- An AKP is then defined in terms of the following parameters:
  - \( N \) - the number of samples.
  - \( T \) - the time interval between each sample.
  - \( K_{\text{min}}, K_{\text{max}} \) - the minimum and maximum stream’s value.
- Given these parameters we can easily transform a stream to an AKP and vice-versa.
**The Architecture**

1. **Pattern Generator**
   - Provides an I/O pattern

2. **Injector**
   - Injects a keystroke stream

3. **Monitor**
   - Reports the I/O stream of each process

4. **Pattern Translator**
   - Reports the I/O pattern
   - Provides an I/O stream

5. **Detector**
   - Reports the I/O pattern

6. **Operating System**
The Correlation Metric

- The **Detection phase** determines the correlation between two AKPs, P and Q.
- We adopt the Pearson Correlation Coefficient (PCC) in order to compare AKPs.

\[ \text{PCC} = \frac{\text{cov}(P, Q)}{\sigma_p \cdot \sigma_q} \]

- A PCC of 0 means no correlation, +1 and -1 direct and inverse correlation.
- Its choice is appealing due to its **linearity**, that is it is scale and location invariant.

\[ \text{PCC}(P, Q) = \text{PCC}(a + Pc, Q) \]

- Immune to data normalization such as encryption or ignored keystrokes.
Generating Patterns

- The **Generation phase** forges a pattern such that:
  - It must **NOT** resemble any pattern exhibited by legitimate processes.
  - It must be easily identifiable in the output.

- We tested the following patterns:
  - **Random** - every sample is generated at random within the range \([0, 1]\).
  - **Random Fixed Range** - a random permutation of uniformly distributed samples.
  - **Sine Wave** - a discrete sine wave oscillating between \(0\) and \(1\).
  - **Impulse** - a pattern composed of alternated \(0\) and \(1\).
  - **Workload Aware** - maximally uncorrelated to the actual workload (see paper).
We tested our approach against a proof-of-concept keylogger to investigate how the AKP’s parameters influence the PCC.

- **Best value:** $K_{\text{max}} = 50$
Parameters Tuning 2/2

- We tested our approach against a proof-of-concept keylogger to investigate how the AKP's parameters influence the PCC.

- **Best value:** $T=250$
False Negatives

- The PCC is a stable metric even against a random padding of the logged data.
False Positives

- We measured the PCC of the adopted patterns against some real workloads.

- **Best value:** \( N=60 \)
Results

Top monitoring free software list - [http://www.keylogger.org](http://www.keylogger.org)

<table>
<thead>
<tr>
<th>Keylogger</th>
<th>Detected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refog Keylogger Free 5.4.1</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Best Free Keylogger (BFK) 1.1</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Iwantssoft Free Keylogger 3.0</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Actual Keylogger 2.3</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Revealer Keylogger Free 1.4</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Virtuoza Free Keylogger 2.0</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Quick Keylogger 3.0.031</td>
<td>N/A</td>
<td>No log produced</td>
</tr>
<tr>
<td>Tesline KidLogger 1.4</td>
<td>N/A</td>
<td>No log produced</td>
</tr>
</tbody>
</table>

**Parameters**

Pattern=RFR, PCC Threshold=0.60, N=60, T=1000, $K_{\text{min}}=1$, $K_{\text{max}}=50$
Conclusions

• We presented an **unprivileged** approach to counter the plague of keyloggers.

• **Effective** against real keyloggers in a realistic scenario.

• The proposed architecture is OS **independent**.

• The resulting tool (GPL licensed) will be soon made public.
Thanks for the attention!
Any questions?