FirewireBlocker

A Software Defense against Firewire-based Physical Security Attacks on Windows Systems

Benjamin Böck
Security Research Lab
Secure Business Austria
bboeck@securityresearch.at

David Huemer
Institute of Software Technology and Interactive Systems
Vienna University of Technology
dhuemer@ifs.tuwien.ac.at

V 1.0, 2009-08-12

Latest version at
http://www.securityresearch.at/publications/windows_firewire_blocker.pdf

1 Overview

In this paper we present a software solution to Firewire-based physical security attacks on Microsoft Windows operating systems. While the problem is described in section 2, our solution is presented in section 3, which also gives details on the concrete implementation and decisions taken in the course of development. Section 4 (page 4) describes the positive testing results and observations made. Finally, section 5 (page 5) summarizes the results and provides suggestions for further research.

2 Problem

As outlined in our paper [01], modern operating systems (including Windows 7) seem susceptible to Firewire-based physical security attacks. As Firewire allows full read/write memory access via Direct Memory Access (DMA), running systems can be cracked if an attacker has access to an active Firewire port. As a result, attackers can gain full access to running systems. For example, the Windows password check can be bypassed, even if the screen is locked or the system is brought back from standby mode. Moreover, encryption solutions like Microsoft Encrypted File System (EFS) and even full disk encryption solutions like Microsoft BitLocker **cannot** protect against such attacks in most cases, if the system is running and the boot phase has finished [01].

2.1 Threat Scenario

It is unlikely that an adversary is able to attack your system via the Firewire port while you are sitting in front of it – at least without you noticing it. The threat scenario we want to cover is thus the following: While you are away from your system but it is either *running* **and** *screen-locked* **or** in *standby mode*, an adversary sneaks up to the machine and uses an active Firewire port to conduct the attack. Systems without a built-in Firewire interface are at risk as well as PCMCIA/CardBus interfaces (e.g., PCMCIA slot) on the system would allow on-the-fly installation of Firewire interfaces.

This applies for situations where full disk encryption is implemented. **Without** full disk encryption, the system is **vulnerable in any event** if it can be placed into a **running state** with an **enabled Firewire** port.

Thus, the following system configurations and states have to be distinguished and are either *covered* or *not covered* by our solution.

Windows State	Full disk encryption	
	With Pre-boot Authentication	**Without Pre-boot Authentication**
Running/locked	Covered	Covered
Standby	Covered	Covered
Powered off	Not covered[A]	Covered
Hibernated	Not covered[A]	Covered
Running/unlocked	Not covered[B]	Not covered[B]

[A] In these scenarios, pre-boot authentication protects against Firewire-based physical attacks.

[B] If an attacker is able to gain physical access to an unlocked, running system with a logged-on user, all is lost. Our solution cannot protect against acts of negligence.

3 Solution and Implementation

Our **FirewireBlocker** software [02] establishes a simple yet effective defense against Firewire-based physical security attacks on unattended systems. The basic idea is to automatically disable Firewire interfaces whenever the screen is locked or users log off and re-enable them as soon as they return.

The solution is implemented as a .NET Windows service running with SYSTEM privileges. Two basic programmatic problems had to be solved:

- Getting noticed of system logon/display lock/screensaver events;
- Disabling and re-enabling Firewire and PCMCIA/CardBus controllers (as Firewire interfaces could be added on-the-fly using the latter).

3.1 Event Notification

One of the requirements for our solution is receiving notification of events like:

- **Logon:** Occurs when a user logs on to Windows;
- **Logoff:** Occurs when a user logs off from Windows;
- **DisplayLock:** Occurs when a user locks the screen;
- **DisplayUnlock:** Occurs when a user unlocks the screen;

- **StartScreenSaver:** Occurs when screen saver starts;
- **StopScreenSaver:** Occurs when screen saver stops.

The following sections describe different methods to reach this goal along with specific pros and cons. We ultimately settled for the System Event Notification Service (SENS), as described in section 3.1.4 System Event Notification Service (SENS), page 3.

3.1.1 .NET SystemEvents

.NET 2.0+ provides the SystemEvents.SessionSwitch event [03] which "occurs when the currently logged-in user has changed". However, the event is realized as a listener to window messages. As services do not have message loops, they cannot receive window messages – thus, these kind of events cannot be used in services (no event is ever raised).

Developers can work around the limitation by making the service interactive (allowing access to the desktop) and including an invisible window to receive the messages. This method is described by Microsoft in [04]. However, such a design introduces significant risk for privilege escalation attacks, if the concerned service is running with high privileges. In the past, for example, such design has been exploitable through shatter attacks [05]. Luckily, the inherent security risk is also taken into account by Windows Vista, which presents annoying nag screens whenever a service tries to create a window. One might be able to circumvent this warning message by moving the window to another window station but for aforementioned security reasons we did not try to do so.

3.1.2 WMI Syslog Monitoring

Another "solution" often mentioned online (for example, [06]) is to turn on logon auditing via policy and then use the ManagementEventWatcher and WMI (Windows Management Instrumentation) to receive notification of said events. Obviously, such a solution is not generic enough for universal application, as prerequisites (turning on logon auditing) are required.

3.1.3 Windows Notification Packages

"In pre-Windows Vista versions of Windows, Winlogon notification packages are registered DLLs that the Winlogon process loads. These DLLs receive Winlogon notifications and handle different Winlogon events." [07]

As our solution aims to also work on current versions of Windows Vista and Windows 7, Windows Notification Packages could not be considered.

3.1.4 System Event Notification Service (SENS)

According to Microsoft [07], most events accessible via Windows Notification Packages are also usable via the System Event Notification Service (SENS), which is also available on more recent versions like Windows Vista and Windows 7 and thus represents an adequate technique for our solution. It is available on Windows 2000 and later. The ISensLogon interface is described in [08].

3.2 Enabling/Disabling Hardware

Besides receiving notification on Logon-events, the other problem to be solved was programmatic enabling/disabling of hardware devices. A quick search yielded Microsoft's **DevCon** command line

utility. According to Microsoft, source code for DevCon is also available in the Windows DDK [09], which has been superseded by the Windows Driver Kit (WDK) [10].

By accessing SetupAPI via .NET, hardware devices can be listed and enabled/disabled programmatically. Device classes are denoted by unique GUIDs. Several "System-Defined Device Setup Classes" exist, as described by Microsoft. The two device classes relevant to our problem are [11]:

- IEEE 1394 Host Bus Controller
  - Class = 1394
  - ClassGuid = {6bdd1fc1-810f-11d0-bec7-08002be2092f}
  - This class includes 1394 host controllers connected on a PCI bus, but not 1394 peripherals. Drivers for this class are system-supplied.
- PCMCIA Adapters
  - Class = PCMCIA
  - ClassGuid = {4d36e977-e325-11ce-bfc1-08002be10318}
  - This class includes PCMCIA and CardBus host controllers, but not PCMCIA or CardBus peripherals. Drivers for this class are system-supplied.

4 Testing

To test our solution, we conducted Firewire-based physical security attacks on a Windows 7 RTM (32 and 64 bit) and on a Windows Vista system, as described in our paper [01]. Without our application, the attacks succeeded as anticipated.

With running FirewireBlocker service, Firewire and PCMCIA/CardBus controllers were automatically turned off as soon as the PC was locked, a user logged off or the screensaver started. Controllers were automatically enabled again when the complementary events occurred. Trying to abuse a Firewire port of a system protected this way led to an error on the attacker side; attacks did not succeed.

Our defense software thus successfully protects against Firewire-based physical security attacks on running, unattended systems.

4.1 Handling already disabled Devices

One problem coming to mind is how to handle devices which have already been disabled intentionally. These devices should not be enabled automatically because they most likely were disabled for a reason. In order to cope with this problem, we thought of temporarily storing the original state of devices before disabling them so already disabled devices are left out during the re-enabling phase.

However, on our test systems, we noticed that devices which have originally been disabled, then programmatically "disabled again" and finally enabled through our application redeemed their original (disabled) status. Interestingly, the effect even outlasts a reboot.

Further testing showed that we could only programmatically enable devices we had disabled the same way. Devices disabled manually could not be enabled programmatically with our application.

Though this result is actually in favor of our intentions and saves us from additional programming work, it is subject to further research to investigate the cause and universal validity of the described behavior.

4.2 StartScreenSaver and DisplayLock

While Windows 9x allowed for a separate screen saver password, Windows 2000 and up use the standard Windows Logon information dialog when checking the "On resume, password protect" option. We thus assumed the system would automatically trigger the Logoff event when the screensaver starts. However, in the course of our tests it showed that the Logoff event occurs when the screensaver is turned off. We thus had to monitor screensaver events as well, in order to close the possible attack window during the time the screensaver is running.

5 Conclusion and Future Work

In this paper we present a simple yet effective software defense against hardware Firewire-based physical security attacks on Windows systems. Using our approach, Firewire and PCMCIA/CardBus interfaces which could potentially allow Direct Memory Access (DMA), are only enabled while users are actively working with a system, that is, a user is logged on, the screen unlocked and no screensaver running. Adversaries targeting the system cannot conduct Firewire-based physical security attacks while the system is locked. Our implementation is lightweight, easy to install and maintenance free, as the Windows service can keep running in the background, likely without users ever noticing.

The solution should be checked for possible side effects, further testing on different systems is required. For example, it is unclear why manually disabled devices cannot be enabled programmatically through our solution.

By now, we do not automatically disable devices when our service starts (normally after system boot). Theoretically, this leaves the following attack window: an attacker reaching an unattended, running system (Firewire devices enabled), pulls the plug (shutting down would lead our service to disable the devices) and then reboots the system, with the devices staying enabled. However, if a system is running with Firewire ports enabled, an attacker could have gone for a Firewire-based physical attack directly. We thus do not recognize additional risk in not disabling Firewire devices on start of our service. We may change our design in future versions, if required.

Problems may arise where Firewire and/or PCMCIA/Cardbus components should be able to operate while the system is in a locked state. For example, PCMCIA/CardBus network cards lose connection automatically, as soon as the screen is locked or the user logs off. On the other hand, hard drives connected via Firewire will prevent the service from disabling the Firewire controller if a copy operation is in progress. Future versions should try to meet such requirements.

In this first proof-of-concept, the FirewireBlocker service is running with SYSTEM privileges in order to be able to enable/disable hardware. While users can normally not interface with our service, risk for privilege escalation remains. For example, if users with normal user rights have write access on the executable, they could replace it with a malign piece of software which would then be started with SYSTEM privileges. Future versions should establish the principle of least privilege. Further research is required to identify the fewest required privileges the FirewireBlocker service has to run with in order to be able to serve its purpose.

6 Annex

6.1 About the Authors

Benjamin Böck is employed at Secure Business Austria, an industrial research center for IT-Security based in Vienna/Austria and is also an independent information security consultant. He holds several IT-related master's degrees from Vienna University of Technology where he is currently a Ph.D. student. Benjamin lectures on information security related topics at the Vienna University of Technology and several universities of applied sciences; he is also an instructor for professional penetration testing courses. He has a background in IT auditing from one of the Big Four and also explored other information security related domains such as digital forensics. His main interests are in the area of penetration testing with a focus on web applications.

David Huemer is employed at Vienna University of Technology and Secure Business Austria. His topics of interest are Auditing, Penetration Testing (Web, Application, Operating System, Network), Grid and Cloud Computing Security and Linux/Unix. He holds a master's degree in Business Informatics from Vienna University of Technology and is CISSP, CEH and LPIC-1 certified. Currently he is finishing his Ph.D. in the area of Grid Computing Security and his master's degree in Software Engineering and Internet Computing, both at Vienna University of Technology.

6.2 About the Security Research Lab

Members of the Security Research Lab conduct security research on various subject areas of information security. It is situated in Vienna, Austria.

For more information please refer to http://www.securityresearch.at

Security Research is a strategic partner of Secure Business Austria, an industrial research center for IT-Security founded by the Vienna University of Technology, Graz University of Technology and University of Vienna. http://www.sba-research.org/

6.3 Acknowledgement

The work described in this paper has partially been supported by Secure Business Austria and Vienna University of Technology.

7 References

[01] **Benjamin Böck.** Firewire-based Physical Security Attacks on Windows 7, EFS and BitLocker. http://www.securityresearch.at/publications/windows7_firewire_physical_attacks.pdf (accessed August 13, 2009)

[02] **FirewireBlocker (software).** http://www.securityresearch.at/publications/firewireblocker.zip (accessed August 13, 2009)

[03] **Microsoft MSDN**. SystemEvents.SessionSwitch Event. http://msdn.microsoft.com/en-us/library/microsoft.win32.systemevents.sessionswitch.aspx (accessed August 13, 2009)

[04] **Microsoft MSDN**. SystemEvents Class. Includes code example that shows how to handle system events by using a hidden form in a Windows service. http://msdn.microsoft.com/en-us/library/microsoft.win32.systemevents.aspx (accessed August 13, 2009)

[05] **Adrian Leuenberger**. Shatter Attack. Privilege Escalation on Win32 Systems. http://www.csnc.ch/misc/files/publications/ShatterAttack_CSNC.pdf (accessed August 13, 2009)

[06] **Microsoft MSDN Visual C# Developer Center**. Detecting User logon event. http://social.msdn.microsoft.com/forums/en-US/csharpgeneral/thread/9fd8bd47-82b5-4283-8a73-da2c8631fef8 (accessed August 13, 2009)

[07] **Microsoft Technet**. Winlogon Notification Packages Removed: Impact on Windows Vista Planning and Deployment. http://technet.microsoft.com/en-us/library/cc721961(WS.10).aspx (accessed August 13, 2009)

[08] **Microsoft MSDN**. ISensLogon Interface. http://msdn.microsoft.com/en-us/library/aa376860(VS.85).aspx (accessed August 13, 2009)

[09] **Microsoft Support**. The DevCon command-line utility functions as an alternative to Device Manager. http://support.microsoft.com/kb/311272/en-us (accessed August 13, 2009)

[10] **Microsoft Windows Hardware Developer Central.** Windows Driver Kit (WDK). http://www.microsoft.com/whdc/Devtools/wdk/default.mspx (accessed August 13, 2009)

[11] **Microsoft MSDN**. System-Supplied Device Setup Classes. http://msdn.microsoft.com/en-us/library/ms791134.aspx (accessed August 13, 2009)