Chapter 10

IMPLEMENTING BOOT CONTROL FOR WINDOWS VISTA

Yuki Ashino, Keisuke Fujita, Maiko Furusawa, Tetsutaro Uehara and Ryoichi Sasaki

Abstract A digital forensic logging system must prevent the booting of unauthorized programs and the modification of evidence. Our previous research developed Dig-Force2, a boot control system for Windows XP platforms that employs API hooking and a trusted platform module. However, Dig-Force2 cannot be used for Windows Vista systems because the hooked API cannot monitor booting programs in user accounts. This paper describes an enhanced version of Dig-Force2, which uses a TPM and a white list to provide boot control functionality for Windows Vista systems. In addition, the paper presents the results of security and performance evaluations of the boot control system.

Keywords: Evidence integrity, boot control, Windows Vista

1. Introduction

Personal computers are often the instruments and/or victims of electronic crime. This makes it important to securely log and store all potential evidence for use in legal proceedings [5]. The logging system should operate in a "sterile" environment, log and store all operational data and enable a third party to verify the integrity of the logged data.

We previously designed the Dig-Force system [1] to address these issues. Dig-Force reliably records data pertaining to computer usage on the computer itself and uses chained signatures to maintain the integrity of the evidentiary data. Dig-Force has been shown to be effective even on standalone computers located outside a protected network. Maintaining the security of Dig-Force requires an environment that prevents the execution of boot jamming programs as well as the modification of the Dig-Force program itself.

Our next version, Dig-Force2 [2], was developed to maintain a secure environment under Windows XP. Dig-Force2 implements boot control using API hooking and incorporates a trusted platform module (TPM) [7] to prevent boot jamming programs from executing and to detect modifications to Dig-Force. Dig-Force2 runs as a Windows service and hooks the `RtlCreateProcessParameters` API [3]. It verifies that any booting program that executes is non-malicious using a white list and TPM. Only a booting program on the white list is allowed to execute.

Unfortunately, Dig-Force2 does not operate on Windows Vista because the `RtlCreateProcessParameters` API hook cannot be used to monitor booting programs in user accounts. This paper describes an enhanced version of the boot control system, named Boot Control Function for Windows Vista (BCF/Vista). The enhanced system uses a TPM and white list with a controller process that runs as a Windows service, along with an agent process that executes within the user account.

2. Dig-Force2 Boot Control

Dig-Force2 [2] is designed to be used by administrators, users and system verifiers. Administrators are responsible for setting up and configuring computer systems. Users operate computers using their assigned Windows XP accounts, which are referred to as "user accounts." System verifiers are responsible for verifying the log files created by the system. An administrator can also serve as a system verifier.

Figure 1 presents the Dig-Force2 architecture. It has five components: (i) logging module, (ii) storage module, (iii) boot control function, (iv) Windows service, and (v) user account. The logging module captures data about computer operations (e.g., user actions and system behavior) and sends it to the storage module. The storage module tags the received data with the date, time and data type. Additionally, it digitally signs the data with a hysteresis signature [4, 6] using a public key stored in the TPM before writing the data to the log file. The hysteresis signature ensures that any alterations to the data in the log file are detected [1].

It is essential that boot jamming programs are prevented from executing and that the logging and storage modules are not modified in an unauthorized manner. In order to maintain such a secure environment, Dig-Force2 hooks the `RtlCreateProcessParameters` API, which is invoked whenever a Windows XP program starts. Dig-Force2 then checks that the program is on the white list before permitting it to execute. The check involves computing a hash value of the program and comparing the value with the matching digital signature from the white list after decrypting it using a public key stored in the TPM.

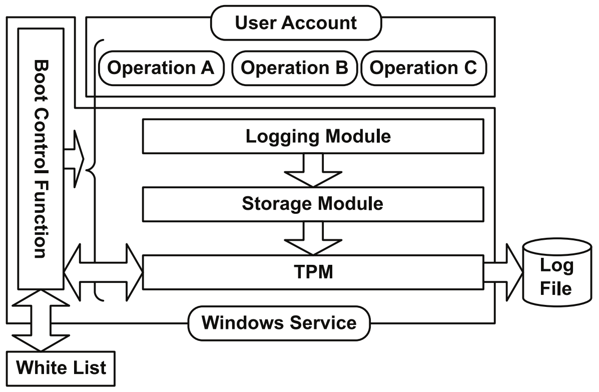

Figure 1. Dig-Force2 architecture.

3. Boot Control Function for Windows Vista

Dig-Force2 runs as a Windows service under Windows XP, which means that it cannot be stopped by anyone except the administrator. However, in Windows Vista, the hooking program must run under a user account if Dig-Force2 is to hook programs executing in a user account. This means that the user can stop the hooking program at any time using the Windows Task Manager.

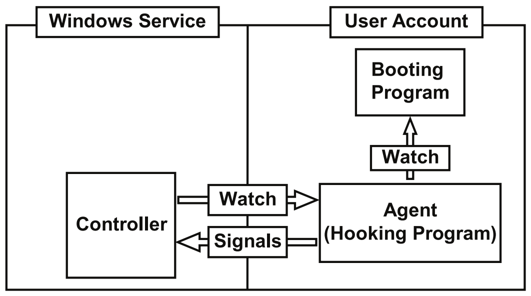

Figure 2. Boot Control Function (Windows Vista) architecture.

Figure 2 presents the architecture of the Boot Control Function for Windows Vista (BCF/Vista). The agent is an enhanced hooking program that executes in the user account; it monitors all booting programs executed from the user account and communicates their condition to the controller. The controller, which is implemented as a Windows service, prevents a user from terminating the agent.

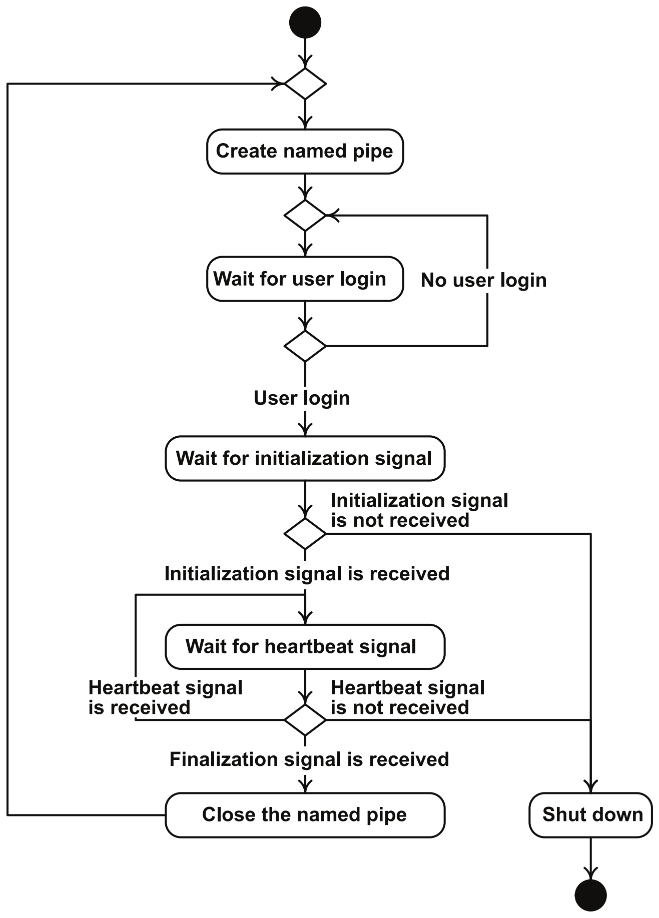
Create named pipe
Wait for user login
No user login
User login
Wait for initialization signal
Initialization signal is not received
Initialization signal is received
Wait for heartbeat signal
Heartbeat signal is received
Heartbeat signal is not received
Finalization signal is received
Close the named pipe
Shut down

Figure 3. Controller logic flow diagram.

The remainder of this section describes the logic of the controller and agent, and outlines the procedures that must be followed by an administrator to set up BCF/Vista.

3.1 Controller

The controller is an administrator-level Windows service, which ensures that the agent hooking program is always running. Figure 3 presents the controller logic, which involves four main steps. First, the controller creates a named pipe to communicate with an agent executing in a user account. It then waits for an initialization signal from the agent which indicates that a user has logged on. If an initialization signal from

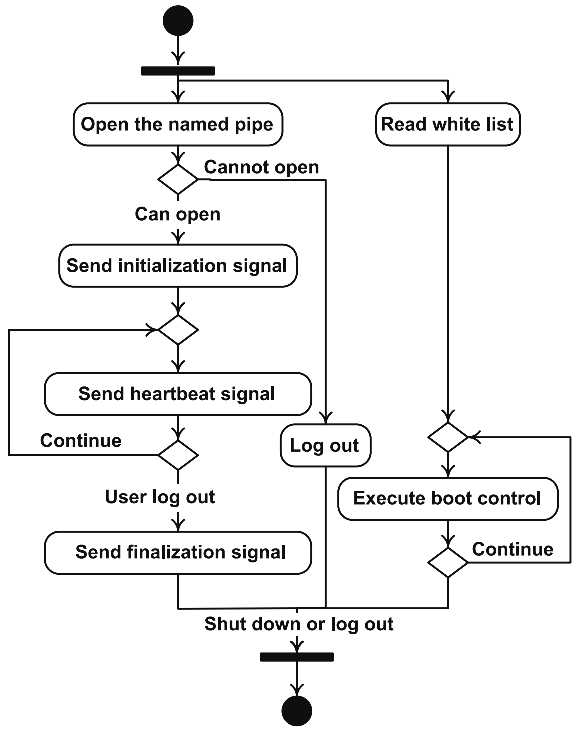

Figure 4. Agent logic flow diagram.

the agent is not received by the controller within a specified time interval, the controller shuts down Vista. The controller then listens for a heartbeat signal from the agent, which indicates that the agent is executing. If the heartbeat signal is not received by the controller, Vista is shut down. When the user logs off, the controller receives a finalization signal from the agent, upon which it closes the named pipe.

3.2 Agent

The agent operates in a manner similar to Dig-Force2 except that the `CreateProcessW` and `CreateProcessA` APIs are hooked instead of `RtlCreateProcessParameters`. However, the BCF/Vista agent runs in a user account to enable boot process hooking unlike Dig-Force2, which runs as a Windows service.

Figure 4 presents the boot control logic of the agent. The agent first opens the named pipe created by the controller. If the agent cannot open the pipe, the agent forcibly logs out the user. Next, the agent sends an initialization signal to the controller, reads the white list and hooks the `CreateProcessW` and `CreateProcessA` APIs. This becomes a secondary process that compares the hash value of the booting program with the white list value as in the case of Dig-Force2. The agent then sends the controller a heartbeat signal at a predefined interval. If the user shuts down the computer or logs out, the agent sends a finalization signal to the controller.

3.3 System Configuration

Before a user can access the computer, the administrator must configure it using the following procedure:

- **Step 1:** Add a standard user account for the user.
- **Step 2:** In order to start the agent, add the task to the task scheduler.
- **Step 3:** Set the program permissions to "read only" from the user account; this prevents the user from modifying programs.
- **Step 4:** Set the permission for "Start-Up" in the user account to "read only" to prevent the user from adding start-up programs.
- **Step 5:** Check "Audit Process Tracking" in the local security policy.
- **Step 6:** Install the white list.
- **Step 7:** Add the controller as a Windows service.
- **Step 8:** Enable BitLocker.
- **Step 9:** Set the BIOS password.
- **Step 10:** In the safe mode, enable the booting service (Step 7).

The administrator then sets up the task scheduler as follows:

- **Step 1:** Create a new task schedule in the folder Task Scheduler Library of the task scheduler.
- **Step 2:** Set "When running the task, use the following user account:" to point to the user account.
- **Step 3:** Set "At log on" and "On an event" with target log to "Security" when triggered.

- **Step 4:** Set "Start a program" to the status of "Action" and register the program path of the agent.

- **Step 5:** Uncheck the checkbox of "Start the task only if the computer is on AC power."

4. System Evaluation

We consider four attacks where the attacker is a user with a standard user account: (i) BCF/Vista removal or modification, (ii) white list modification, (iii) BCF/Vista start-up blocking, and (iv) jamming.

In order to remove or modify the BCF/Vista program file, an attacker must have administrator privileges, which is not possible without the administrator's password. Alternatively, the attacker could attempt to access the hard drive directly, for example, by booting another OS from an alternative storage media or installing the hard drive in another machine. However, the BIOS of the computer is set to boot only from the hard drive, which is fully encrypted using BitLocker. These security measures prevent access to the BCF/Vista program files.

The goal of an attack on the white list is to add a program to the list. To do this, the attacker must be able to add the digital signature of the program to the white list. This requires the secret key used to create the list or the creation of a fabricated white list with a fake key pair. But these will not work because the attacker neither has the secret key nor the password required to install a fake public key in the TPM.

The third attack, blocking the booting of BCF/Vista, is not possible because a Windows service cannot be modified without the administrator's password. This password is not available to the user.

A boot jamming attack is effective only if the jamming program starts before `CreateProcessW` and `CreateProcessA` are hooked. Since the agent and the hooking start automatically at login, the attacker would have to add the jamming program to the start-up list. This is not possible because the start-up directory permission level is set to "read only."

The evaluation of BCF/Vista was conducted using a Dell VOSTRO 1310 with a 2.5 GHz Intel Core2 Duo T9300 CPU and 4 GB RAM running Windows Vista Ultimate Edition. The program required 1,080 seconds to create a white list by calculating the hash values and the digital signatures for the 3,273 `.exe` files residing on the machine. The signing process clearly requires a considerable amount of time. However, it is performed only once when the white list is created and, therefore, does not impact normal computer operations.

Table 1. Booting times.

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
`notepad.exe`	1.5491	0.0105	0.0104	0.0105	0.0104
`calc.exe`	1.4850	0.0124	0.0126	0.0124	0.0125
Microsoft Word 2007	1.6446	0.0129	0.0127	0.0132	0.0127
Microsoft PowerPoint 2007	1.5849	0.0166	0.0169	0.0165	0.0169

Table 1 shows the time taken to boot four programs, where each program was booted five times in succession (Trials 1–5). The boot period of a program is measured from the time the boot control function permits the program to boot to the time when `CreateProcessW` and `CreateProcessA` are called.

Note that a significant amount of time (1.4850 to 1.6446 seconds) is required for booting during the first trial when BCF/Vista has to read the white list. After this, the booting time is much less because BCF/Vista uses the white list data, which was read the first time it was executed. The booting time is, thus, acceptable and BCF/Vista has negligible impact on program operation.

5. Conclusions

BCF/Vista provides a secure and reliable environment for logging data pertaining to computer operations. In particular, it preserves the integrity of evidence by preventing the booting of unauthorized programs and evidence modification. The architecture, which uses special controller and agent processes, a TPM and a white list to provide boot control functionality for Windows Vista systems, has a negligible impact on system performance.

References

[1] Y. Ashino and R. Sasaki, Proposal of digital forensic system using security device and hysteresis signature, *Proceedings of the Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 3–7, 2007.

[2] K. Fujita, Y. Ashino, T. Uehara and R. Sasaki, Using boot control to preserve the integrity of evidence, in *Advances in Digital Forensics IV*, I. Ray and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 61–74, 2008.

[3] Microsoft Corporation, Services, Redmond, Washington (msdn .microsoft.com/en-us/library/ms685141.aspx).

[4] K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki and H. Yoshiura, Digital document sanitizing problem, *Institute of Electronics, Information and Communication Engineers Technical Reports*, vol. 103(195), pp. 61–67, 2003.

[5] R. Sasaki, Y. Ashino and T. Masubuchi, A trial for systematization of digital forensics and proposal on the required technologies, *Japanese Society of Security Management Magazine*, April 2006.

[6] S. Susaki and T. Matsumoto, Alibi establishment for electronic signatures, *Transactions of the Information Processing Society of Japan*, vol. 43(8), pp. 2381–2393, 2008.

[7] Trusted Computing Group, Beaverton, Oregon (www.trustedcom putinggroup.org).