Update State Tampering: A Novel Adversary Post-Compromise Technique on Cyber Threats

Sung-Jin Kim[1], Byung-Joon Kim[1], Hyoung-Chun Kim[1] and Dong Hoon Lee[2]

[1] National Security Research Institute of South Korea, P.O.Box 1, Yuseong, Daejeon, Republic of Korea
ksj1230@nsr.re.kr, bjkim@nsr.re.kr, khche@nsr.re.kr

[2] Graduate School of Information Security, Korea University, Seoul, Republic of Korea
donghlee@korea.ac.kr

Abstract. In modern cyber threats, an attacker should gain persistency at target systems to achieve attack objectives. Once an attacker's 0-day vulnerabilities on target systems are patched, the attacker may lose control over the system. However, what happens when the attacker manipulates the component resources on a Windows system? We found the ways to generate known but invisible vulnerabilities on a victim's system. Our findings are as follows; first, we have found ways to replace a component to an old vulnerable version while maintaining the current update records; second, we also found that the Windows system does not recognize the replaced components. We define the first issue as a package-component mismatch and the second issue as blind spots on the Windows update management. They have been identified on all version of Vista and later, including desktop platforms and server platforms. Based on our findings above, we propose Update State Tampering technique to make invisible security holes on target systems. Meanwhile, we also offer corresponding countermeasures to detect and correct the package-component mismatch. In this paper, we introduce the problems of the current Windows update management, Update State Tampering technique from the attacker's point of view and Update State Check scheme that detects and recovers the package-component mismatch. We stress that Update State Check scheme should be deployed immediately to mitigate large-scale exploitation of the proposed technique.

Keywords: Post-Compromise, Windows Update, Cyber Threat

1 Introduction

In modern cyber threats [3-5], the post-compromise phase [1-2, 6] comes after exploitations on victim's system. At this phase, an attacker should gain persistency on target systems while maintaining the stealth. ATT&CK matrix [7] provided by MITRE introduces 38 techniques of a persistence tactic and 40 techniques of a defense evasion tactic that were used in various APT attack campaigns at the post-compromise phase. In the persistence tactic, an attacker can use the "Accessibility Features" technique [8] to launch a malicious program before a user logged in. For example, by modifying a specific registry key, an attacker can allow a debugger program to be executed with SYSTEM privileges at the login screen [9]. In the defense evasion tactic, APT attack campaigns used the “Modify Registry” technique [10] to hide information within registry keys or to neutralize the UAC. For example, the Regin [11] hides his malware payloads in the registry. The CHOPSTICK [12], used by APT28, stores his configuration block in the registry with the RC4 encryption. And the Shamoon [13] changes a specific registry value to disable remote restrictions on target systems. The techniques in the persistent tactic show how to perform malicious behavior even after a system update or reboot. Except for some rootkit attacks, such as hooking or bootkit, these techniques fall into the misconfiguration or abusing category.

On the contrary, our proposed technique is based on the structural problems of the update management mechanism of Windows platforms. We modify registry values related to component resources and the pending.xml file to replace target components with old vulnerable versions. After things going, the known vulnerabilities of those components will be revived. And target system’s defense mechanisms cannot detect the revived vulnerabilities. Finally, our technique allows an attacker to exploit the hidden vulnerabilities even if the target system is fully patched and the attacker's initial vulnerability is removed.

Package-Component Mismatch. In Microsoft Windows, a component is the smallest unit of a software module. When security vulnerabilities are reported, Microsoft deploys update packages that include some higher version components. When users install the update, the Windows Installer saves the new versions of the component in the component store and projects them to the appropriate locations. On the surface, the process appears simple; however, behind the scenes are complicated registry settings that are not well known. We focus on the fact that the Windows update relies on specific registry settings to determine the current configuration of system components. If an attacker intentionally alters the version names of target components in the registry during the system update, the target components will be rolled back to the old vulnerable versions, or the system may not update the target components. At this time, the installation history of the update packages is not changed. We define this problem as a package-component mismatch. In this paper, we propose three update state tampering methods which cause the package-component mismatches. The identified methods are affected by component dependency issues. We address the issues in Section 8.

Blind Spot Issue. To make matters worse, Windows does not provide a means to detect and recover from the package-component mismatch. Since the replaced components are normal Windows binaries, they are loaded without any alert to the system. Built-in tools such as; system file checker (SFC) [14] and Windows Update Check [15] cannot verify the package-component mismatches. Further, security compliance tools, such as Microsoft baseline security analyzer (MBSA) [16], also do not check the consistency between update packages and components installed in the system. We call these issue blind spots on the Windows update management. In this paper, we introduce the impact of blind spot issue on both server platforms and desktop platforms.

Assumptions and Attack Scenario. The proposed technique requires administrator privileges to modify specific component resources. Therefore, we recommend that an attacker exploits a known vulnerability or a 0-day on the target system, including local privilege escalations, before using the proposed technique. We also present an attack scenario that re-infects the target system after the attacker's initial exploit has been removed. In our scenario, we assume that only the attacker's initial exploit and its payloads are removed when the target system is restored. This scenario does not include the case where all system components are re-installed in the recovery phase. Also, the attacker must be in a position to gain access to vulnerable service ports on the target system for re-infections.

Contributions. The following contributions are made in this paper:

- *Problems in Windows update management*
  To the best of our knowledge, the package-component mismatch and the blind spot issue are the first structural problems in the Windows update management mechanism that have not been reported. We outline the concept of a package-component mismatch and two types of blind spots of Windows update management and reveal the potential threats associated with them. The blind spot issue occurs because Windows does not care about the consistency between update packages and components installed on the system. These issues affect all server and desktop platforms.
- *Update state tampering technique*
  We outline three update state tampering methods and a toy example. The toy example describes how an attacker can generate the package-component mismatch on a target system. Finally, we show that the vulnerability lifecycle on a local system can be extended by exploiting the blind spot issue.
- *Countermeasures to eliminate the blind spots*
  The blind spot issue can be resolved by mutually verifying the package information and component information. We could extract package-component mapping information from the registry settings scattered inside a local. We also propose update state check scheme, which detects and corrects package-component mismatches by using the package-component mapping information.

Outline. The remainder of this paper is structured as follows: In Section 2, we introduce the background on Windows update management. In Section 3, we propose the fundamental problems of the current Windows update management mechanisms. In Section 4, we propose the Update State Tampering technique and an attack scenario. In Section 5, we measure the impact of the Update State Tampering on several different versions of Windows platforms. In Section 6, we visit the previous studies whether they can detect tampered components. In Section 7, we provide the Update State Check scheme as countermeasures. In Section 8, we summarize features of the update state tampering technique and mitigations and then conclude our work in Section 9.

2 Background

In this section, we provide the background on Component-Based Servicing (CBS) [17-18] and pending file rename operations [19] that are governing Windows update. And then we introduce the fundamental problems of the current Windows update management mechanisms. We explain the details based on Windows 7 64 bits.

Component-Based Servicing. At the start of installation of the update package, TrustedInstaller.exe, the CBS agent, runs the component copy and registry settings. CBS controls the entire installation and deletion of the Windows components. Within a local system, component resources managed by CBS span a vast area. Among them, we focus on the specific component resources to explain the blind spot issue. Each resource notation is as shown in Table 1.

All component files in a local system are hard-linked from the *CS* to the destination path. For example, the "ntoskrnl.exe" file of the component name "amd64_microsoft-windows-os-kernel_31bf3856ad364e35_6.1.601.17514_none_ca56670fcac29ca9" is hard-linked to under the system32 folder. The P_r contains the installed package names. Each sub key of the P_r designates a package name. For example, the sub key "Package_109_for_KB4012215~31bf3856ad364e35~amd64~~6.1.1.2" represents the 109th package included in the rollup security update KB4012215. And the C_n provides the component names that make up the system. For example, the sub key "amd64_microsoft-windows-os-kernel_31bf3856ad364e35_none_2003e93c9e12938" represents the component name that contains the "ntoskrnl.exe" file. The current component version can be found on the registry value in C_v.

Table 1. Component Resources

Resource	Path	notation
Component Store (Files)	\Windows\WinSxS\[*Component Name*]\[*File name*]	*CS*
Registry Key for Package Name	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion \Component Based Servicing\Packages\[*Package Name*]	P_n
Registry Key for Component Name	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion \SideBySide\Winners\[*Component Name*]	C_n
Registry Value for Component Version	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion \SideBySide\Winners\[*Component Name*]\[*Windows Version*]\(default)	C_v

Pending File Rename Operations. After the system shutdown or reboot event, the poqexec.exe process performs the component resource replacement. The pending file rename operations place the replacement of the current memory-mapped file to the point after the reboot event. The TrustedInstaller.exe process generates a pending.xml file when installing an update package. The pending.xml file contains essential information for component resource replacement. The tag name refers to the native API, and the ensuing key-value pairs indicate the arguments referenced by the corresponding API. In Table2, the tag name "HardlinkFile" refers to NtSetInformationFile function with the FileLinkInformation argument and the tag name "SetKeyValue" refers to NtSetValueKey function. The tags T_f and T_v will be performed by the poqexec.exe process. The Windows system considers these set of operations as a transaction. When the execution of the transaction is completed, the installation status of the package is changed from "pending" to "Installed".

Table 2. Tags listed in the pending.xml file when replacing a component

Action	Format	notation
Replace component file	<HardlinkFile source="\SystemRoot\WinSxS\[*Component Name*]\[*File Name*]" destination="\??\C:\[*Destination Path*]\[*File Name*]">	T_f
Replace component version name	<SetKeyValue path="\Registry\Machine\Software\Microsoft\Windows\Current Version\SideBySide\Winners \[*Component Name*]\[*Windows Version*]" name=" " type="0x00000001" encoding="base64" value="[*base64 encoded Version Name*]">	T_v

3 Windows Update Management Problems

In this section, we present the concept of package-component mismatch. And we also present the blind spot issue that system's defense mechanisms cannot detect the result of package-component mismatch.

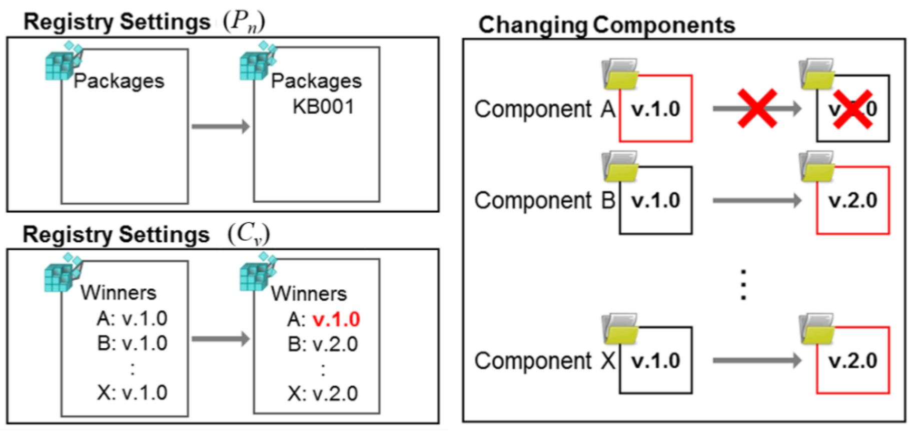

Registry Settings (P_n)
Packages
Packages KB001
Registry Settings (C_v)
Winners A: v.1.0 B: v.1.0 : X: v.1.0
Winners A: v.1.0 B: v.2.0 : X: v.2.0
Changing Components
Component A v.1.0
Component B v.1.0
v.2.0
Component X v.1.0
v.2.0

Fig. 1. Simplified Illustration of a package-component mismatch

3.1 Package-Component Mismatch

The package-component mismatch introduced in this paper refers to a state in which a part of the component is different from the system that is usually updated. The concept of a simple package-component mismatch is illustrated in Fig. 1. It shows the registry and component changes that occur when installing the update KB001 that includes components A through X. When installing KB001, the system adds package information about KB001 to the P_n, and the value of the C_v is updated from v.1.0 to v.2.0. In the end, the poqexec.exe process projects the newer versions from CS to the destination paths with hardlinks. However, during the update procedure of KB001, if an attacker modifies the pending.xml file to set the destination file name as “A_1.0” in T_f. And the component version name as “1.0” in T_v, the poqexec.exe process leaves the component A unchanged. The system works fine even with the package-component mismatch. The package-component mismatch rarely occurs in normal update procedures. However, by obtaining access rights to the component resources, an attacker can manipulate package-component information. If the replaced component is an older version of the component that contains a known vulnerability, the problem becomes more severe.

3.2 Blind Spots on Windows Update Management

Can the package-component mismatch be detected or corrected? As we have seen, the logic to diagnose the package-component mismatch does not exist inside Windows. In this paper, we identify this as a blind spot of the Windows update management mechanism. We have discovered two types of blind spots.

- ***(Type I)*** The system loads the components that do not match the current update state.
- ***(Type II)*** The system does not provide a means for detecting the package-component mismatch.

Blind Spot 1. Code Integrity Policy (Type I). Kernel components must be trustworthy because they can bypass security mechanisms and have unrestricted access to system resources. Windows applies the code integrity policy [20] to verify the trustworthiness of drivers and applications. For example, a 64-bit Windows system enforces Kernel Mode Code Signing (KMCS) [21] policy. During the loading of the kernel component, Windows checks the digital signature signed with the encryption key issued by the certificate authority. This allows Windows to verify the source and integrity of the component. However, checking for package-component consistency is not a role of the code integrity policy. Regardless of the fact that a component has vulnerabilities, every core component that Microsoft provides has a valid digital signature. Therefore, the old components will pass the code integrity mechanism and get loaded into the system without any problems. Further, the system will not leave any warning messages or logs at that time. We diagnose this as the first blind spot.

Blind Spot 2. Windows Update Check (Type II). The Windows Update Agent checks the status of the update packages installed on the system, sends them to the update server, and receives the update packages. The Windows update check performs the following operations:

1. *Collect package information*
   The Windows update check refers to the registry path P_n to determine the packages installed on the system and the installation status.

2. *Download and install packages*
   Based on the information gathered, it identifies the missing updates and downloads the update packages.

From the above operations, the Windows Update Check only references the package information; it does not check the component information. Therefore, even if an attacker changes components and component registry settings, the Windows Update Check cannot detect the package-component mismatch. We diagnose this as the second blind spot.

Blind Spot 3. System file checker (Type II). Windows system administrators can use the "SFC /SCANNOW" command to check for, and repair damaged components. SFC performs the following operations.

1. *Check component information*
   The SFC refers to the registry path C_n and C_v to check a component name and its version name installed on the system.
2. *Verify component damage*
   After confirming the component information, the SFC ensures that the component information obtained from the registry settings matches the component that is hard-linked to the destination path from the *CS*.
3. *Repairs corrupted components*
   If the SFC detects any corruption, it finds the correct version of the component in the component store, then hard-links and reboots.

From the above operations, the SFC only detects and repairs damage to components; it does not take the installed package history into account. Thus, the SFC cannot play any role in detecting the package-component mismatch. We diagnose this as the third blind spot.

4 Update State Tampering Technique

In this section, we introduce a toy example and three update state tampering methods . In each method, we specify the component resources that an attacker can manipulate to cause a package-component mismatch. The goal of the attack is as follows.

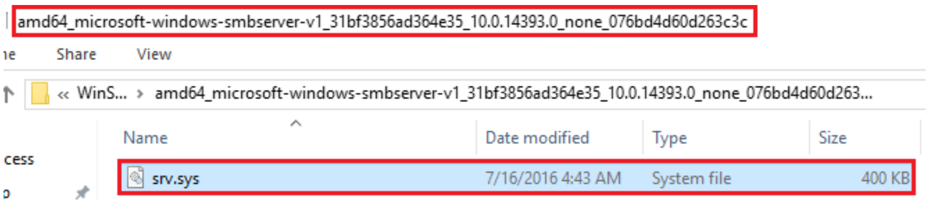
- Replace current components with previous versions that have vulnerabilities while maintaining the record of updates

When this goal is achieved, the system is affected by all the vulnerabilities that the replaced component has. Further, the blind spots make it impossible for the system to detect this as an update status abnormality.

4.1 A Toy Example of Update State Tampering

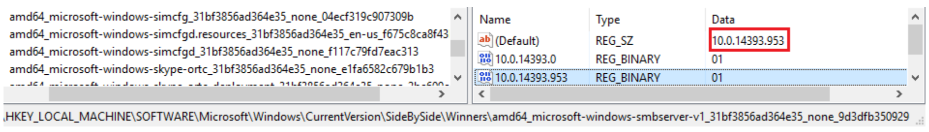
We start with a toy example to make an invisible SMB vulnerability on Windows 10 64 bits. The target components to replace are the srv.sys file and the srv2.sys file with SMB vulnerability known as Eternal Blue [22]. The proposed example assumes that an attacker gains Administrator privileges.

Identify the Target Component. Before tampering with the update state, the attacker should ensure that the system has a vulnerable version of the target component. The Windows system leaves the previous version of the component in the *CS* for recovery, rather than removing it. If the target system has not performed a *CS* cleanup, the vulnerable component will be present inside the *CS*. Fig. 2 shows the target srv.sys file with the SMB vulnerability under the *CS*. The version name of the vulnerable sys.srv file is “10.0.0.14393.0.” Next, we should check the value of C_v. Fig. 3 shows the current version name of the srv.sys file. To downgrade the current srv.sys file, the attacker should replace the value “10.0.0.14393.953” with “10.0.0.14393.0” and then use additional methods.


amd64_microsoft-windows-smbserver-v1_31bf3856ad364e35_10.0.14393.0_none_076bd4d60d263c3c
Share
View
« WinS... › amd64_microsoft-windows-smbserver-v1_31bf3856ad364e35_10.0.14393.0_none_076bd4d60d263...
cess

Name	Date modified	Type	Size
srv.sys	7/16/2016 4:43 AM	System file	400 KB

Fig. 2. srv.sys file that has Eternal Blue vulnerability


```
amd64_microsoft-windows-simcfg_31bf3856ad364e35_none_04ecf319c907309b
amd64_microsoft-windows-simcfgd.resources_31bf3856ad364e35_en-us_f675c8ca8f43
amd64_microsoft-windows-simcfgd_31bf3856ad364e35_none_f117c79fd7eac313
amd64_microsoft-windows-skype-ortc_31bf3856ad364e35_none_e1fa6582c679b1b3
```

Name	Type	Data
(Default)	REG_SZ	10.0.14393.953
10.0.14393.0	REG_BINARY	01
10.0.14393.953	REG_BINARY	01

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Winners\amd64_microsoft-windows-smbserver-v1_31bf3856ad364e35_none_9d3dfb350929
```

Fig. 3. The current version name of the srv.sys file

Use SFC Tool. The System File Checker (SFC) tool detects and recovers component damage based on the version name recorded in C_v. After replacing the value of C_v, the attacker can execute the “SFC /SCANNOW” command with Administrator privileges. The SFC then creates the pending.xml file that downgrades the version of the srv.sys file according to the value of C_v.

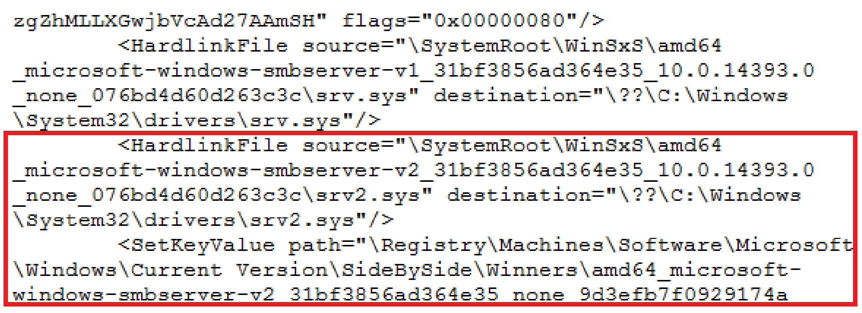

```
zgZhMLLXGwjbVcAd27AAmSH" flags="0x00000080"/>
        <HardlinkFile source="\SystemRoot\WinSxS\amd64
_microsoft-windows-smbserver-v1_31bf3856ad364e35_10.0.14393.0
_none_076bd4d60d263c3c\srv.sys" destination="\??\C:\Windows
\System32\drivers\srv.sys"/>
        <HardlinkFile source="\SystemRoot\WinSxS\amd64
_microsoft-windows-smbserver-v2_31bf3856ad364e35_10.0.14393.0
_none_076bd4d60d263c3c\srv2.sys" destination="\??\C:\Windows
\System32\drivers\srv2.sys"/>
        <SetKeyValue path="\Registry\Machines\Software\Microsoft
\Windows\Current Version\SideBySide\Winners\amd64_microsoft-
windows-smbserver-v2_31bf3856ad364e35_none_9d3efb7f0929174a
```

Fig. 4. Inserted component rollback tasks in the pending.xml file

Tamper at a Single Point. After the pending.xml file is generated, the attacker can register additional component rollback operations in the pending.xml file. Only the TrustedInstller which is one of the NT service accounts can modify the pending.xml file. Therefore, the attacker who gains Administrator privileges must change file ownership, add file modification rights, and then make modifications on the pending.xml file. In this example, we assume that the attacker also downgrades the srv2.sys file. Fig. 4 shows an example of additionally registered component rollback tasks in the pending.xml file. In Fig. 4, the attacker adds the tag T_f with the vulnerable component name "amd64_microsoft-windows-smbserver-v2_31bf3856ad364e35_10.0.14393.0_ none_076104ea0d2e582d," and adds the tag T_r with the vulnerable component version "10.0.0.14393.0." Manipulating pending.xml has several advantages for attackers. It eliminates the need to access and change the scattered component resources individually. Instead, the pending.xml file can be manipulated to let the poqexec.exe process execute arbitrary pending jobs. This means that the tampering point can be minimized to one, single file.

Reboot System. After the system reboot replaced components, srv.sys and srv2.sys are properly loaded into the system due to the blind spot 1. Due to the blind spot 2, the Windows update check reports the same result as the update status before the system is compromised. Now, the hidden Eternal Blue vulnerability is generated. Even if the attacker's initial exploit is removed, the attacker could exploit the remote execution vulnerability in the SMB components to re-compromise the system. In this example, we assume that the target system's SMB service is enabled and that the attacker is in a location accessible to the SMB service port of the target system.

4.2 Update State Tampering Methods

We organized the techniques used in the above toy example by three update state tampering methods.

Method 1. Using SFC to Tamper with the Update State. The SFC detects and recovers corrupted components. However, an attacker can abuse it to tamper with the update state. As stated in Section 3.2, the SFC restores components based on the version information listed in the registry value C_v. The procedure of method 1 is as follows.

1. *Change component version information*
   Access registry path C_v, and then change the target version of a component you wish to replace.
2. *Run the "SFC /SCANNOW" command*
   After the command runs, the system will request a reboot to replace the component. At this point, the pending.xml file will contain a tag T_v with the same component version name in C_v.
3. *Reboot the system*
   The pending jobs are executed to replace the component. The Update Check determines that the package installation is complete. SFC determines that there is no damage to the components. By running the SFC command, attackers can tamper with an update state.

Method 2. Leave Components Unchanged. As stated in Section 4.1, an attacker who gains administrator privileges can arbitrarily add or remove pending jobs in pending.xml. Assume that the attacker removes certain component replacement jobs from pending.xml during package installation. When this occurs, the component will remain unchanged. The procedure of method 2 is as follows.

1. *Delete the target component replacement tag*
   Delete the target component replacement tag listed in the pending.xml file just before rebooting after installing the package. The tag T_f should be deleted to omit the replacement of system files.
2. *Delete the target registry setting tag*
   Delete the target component replacement tag listed in the pending.xml file just before rebooting after installing the package. The tag T_v of a target component should be deleted to omit the operation of target registry settings.
3. *Reboot the system*
   After the system reboot, target components are not replaced, but package information is normally updated. The Update Check determines that the package installation is complete, and the SFC determines that there is no damage to the components.

Method 2 is only applicable if the update creates a pending job for component replacement. The pending job is generated only when there are some changes in kernel components.

Method 3. Revert Components to the Past. During the update package installation, additional tags can be inserted into the pending.xml file to replace components. The procedure of method 3 is as follows.

1. *Insert the target component replacement tag*
   Insert the target component replacement tag listed in the pending.xml file just before rebooting. The tag T_f with a target version name should be added to revert a target component to an old vulnerable version.
2. *Insert the target registry setting tag*
   Insert the target component replacement tag listed in the pending.xml file just before rebooting after installing the package. The tag T_v of a target component should be added to revert the target components to the old vulnerable versions.
3. *Reboot the system*
   After the system reboot, the target components are replaced, and the Update Check determines that the package installation is complete. The SFC determines that there is no damage to the components.

Method 3 is similar to method 2 described above but differs in that it can add component replacement work without limits. When using the methods 2 and 3, the T_f tag that modifies the component file should always be listed together with the T_v tag which changes the version of the associated component.

4.3 Attack Scenario

Based on the update state tampering methods, we introduce an attack scenario.

- Step 1. The attacker uses a zero-day exploit and his malware to launch the first attack on the target system.
- Step 2. One of the update state tampering methods is performed along with the execution of the malware. The replacement target is a previous version of the component that has remote code execution vulnerability.
- Step 3. The system is patched and the attacker's initial exploit is deleted.
- Step 4. The attacker exploits the hidden remote code execution vulnerability to continue malicious activities.

The attacker compromises the target system once with a 0-day exploit in step 1. In step2, the attacker replaces target components with old vulnerable versions. Unlike the method 1, which simply runs the SFC after a component resource C_v manipulation, the method 2 and 3 need to wait for the moment the target system is updated. In step 3, based on the periodic update policy of the target organization, the target system performs an update and the attacker's initial exploit is removed. In step 4, the attacker must be in a position to access the vulnerable service port of the target.

4.4 Extending the Life Cycle of a Vulnerability

We introduce the best time to use the update state tampering technique in the life cycle of vulnerabilities [23, 24]. Suppose an attacker discovered a new vulnerability A, created an exploit and malware, and infected the target system for information-stealing purposes. After a certain amount of time, the software vendors distribute patches. The user tries to recover the infected system by removing the malware and updating vulnerable system components. After this point, the system is no longer affected by the attacker's exploit. This procedure is a typical case of following the vulnerability lifecycle.

What if an attacker tampers with the update status before the target system is recovered? After the recovery procedure, the vulnerability and malware will be removed from the system. However, vulnerable components replaced by the identified methods remain intact. The target system is now subject to known vulnerabilities in those components. If the exposed vulnerabilities contain the remote code execution, the attacker could easily re-compromise the target system. Fig. 5 shows the life cycle of vulnerabilities extended by the update state tampering technique. Typically, the life of a vulnerability A (a 0-day) ends at the patch phase after the discovery-disclosure-exploit phases. However, if an attacker tampers with the update status between the exploit and the patch phases, you can resurrect existing vulnerabilities B and C, and the target system cannot detect the revived vulnerabilities due to the blind spot issue.

In Fig. 5, vulnerabilities B and C are resurrected by the update state tampering and maintained until

1. The next cumulative update replaces the tampered components
2. The operating system of a target host is re-installed
3. The user detects and recovers the tampered components by himself/herself

If only one of the above conditions is met, vulnerabilities B and C are removed. Otherwise, vulnerabilities B and C will remain in the system permanently.

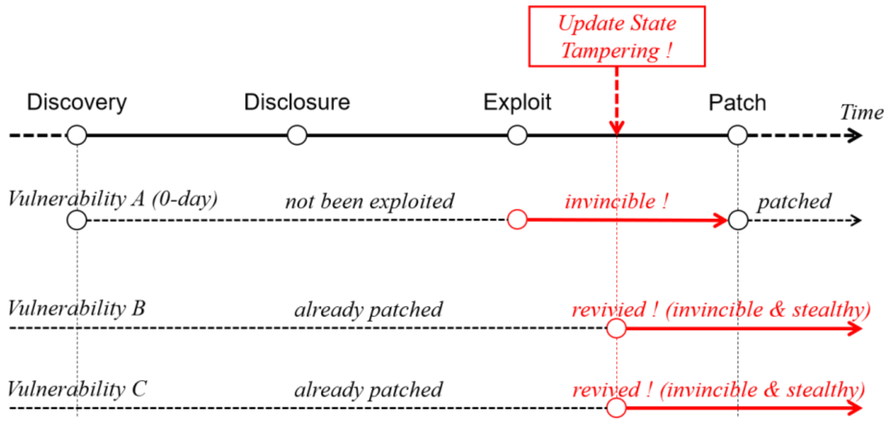


Fig. 5. An illustration of the process of extending a vulnerability life cycle

5 Evaluation

In this section, we evaluate the impact of the identified problems and Update State Tampering methods on different Windows platforms. We applied the latest version of the update to all test platforms in October 2017.

5.1 Impact of the Blind Spots on the Windows Platforms

Since Vista, Windows systems have managed updates based on the CBS. Thus, the presence of blind spots may affect all versions of Windows. Table 3 and Table 4 show the Windows versions that are affected by the outlined blind spots.

Table 3. Impact of blind spots on desktop platforms.

Version	Blind Spot 1	Blind Spot 2	Blind Spot 3
Windows 7	O	O	O
Windows 8	O	O	O
Windows 8.1	O	O	O
Windows 10	O	O	O

Table 4. Impact of blind spots on server platforms.

Version	Blind Spot 1	Blind Spot 2	Blind Spot 3
Windows Server 2008	O	O	O
Windows Server 2012	O	O	O
Windows Server 2012 R2	O	O	O
Windows Server 2016	O	O	O

We have confirmed that the blind spots exist not only on desktop platforms but also on server platforms. Therefore, this issue affects enterprise environments as well as desktop users.

5.2 Impact of the Identified Methods on Windows Platforms

Tables 5 and 6 show the Windows versions that are affected by the identified methods. Windows 8.1 and later do not create a pending.xml file when installing an update; therefore, methods 2 and 3 cannot be applied. This is the same in Windows Server 2012 R2 and later versions. However, method 1 is still valid for all versions of Windows.

All the update state tampering methods described above require the administrative privileges. Therefore, it is recommended to use the proposed methods together with known exploits or 0-days. 0-day attacks are one shot only. Once patched, holes are closed. However, the proposed methods can widen the attack choices. Because of the blind spots, the tampered update state cannot be detected or fixed. The exposed vulnerabilities will persist until the next update to the replaced components.

Table 5. Impact of the identified methods on desktop platforms.

Version	Method 1	Method 2	Method 3
Windows 7	O	O	O
Windows 8	O	O	O
Windows 8.1	O	X	X
Windows 10	O	X	X

Table 6. Impact of the identified methods on server platforms.

Version	Method 1	Method 2	Method 3
Windows Server 2008	O	O	O
Windows Server 2012	O	O	O
Windows Server 2012 R2	O	X	X
Windows Server 2016	O	X	X

6 Previous Work

Before introducing measures to solve the blind spot issue, we checked whether previous studies proposed solutions to remove the blind spots.

Microsoft Baseline Security Analyzer (MBSA). MBSA is a free security compliance tool that detects insecure configurations and missing security updates in the system. This tool provides a report of security vulnerabilities and solutions that exist in the system. However, we looked at how to check for the update state. We used Process Monitor [25] to monitor the behavior of MBSA v.2.3 in 64-bit Windows 7 and checked the results. When MBSA starts a scan, it calls the TrustedInstaller.exe process to test the update status and check the package installation status. The registry paths referenced are shown in Table 7. They are the same registry paths that Windows Update Check verifies. Similarly, MBSA does not play any role in detecting package-component mismatches. We tampered with the components according to the toy example in a test VM and then performed MBSA on the system. As a result, it has been confirmed that the tool cannot detect an update status abnormality.

Table 7. Registry paths frequently referenced by the MBSA and MyPCInspector

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Packages\[*Package Name*]
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\PackagesPending\[*Package Name*]
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\PackageDetect\[*Package Name*]
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\ApplicabilityEvaluationCache\[*Package Name*]

MyPCInspector. MyPCInspector [26] is a third-party security compliance tool used by Korean government offices to check the security status of Windows-based PCs. It provides functionality similar to the MBSA and examines the security settings and security update status of Windows as well as the security settings of commonly used third-party applications.

We used the process monitor tool to observe the behavior of MyPCInspector v.3.0 in 64-bit Windows 7 and checked the results. MyPCInspector calls the TrustedInstaller.exe process to check the update status. The registry paths it references when checking the update status are the same as those in Table 7. Thus, MyPCInspector does not perform any extra work to detect the package-component mismatch. We tampered with the components according to the toy example in a test VM and then performed MyPCInspector on the system. As a result, it has been confirmed that the tool cannot detect an update status abnormality.

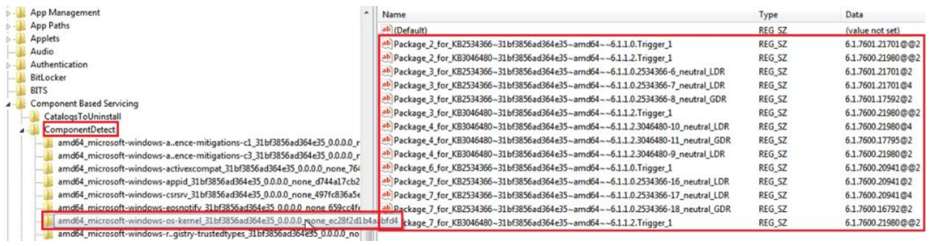
Penetration Testing. Unlike the above tools, Metasploit [27] is a type of penetration testing tool. There are also remote vulnerability scanners such as OpenVAS [28], Greenbone [29], Nessus [30], and Nexpose [31]. Performing pentests is the most obvious way to verify vulnerabilities on the target systems. The scanning result will not be able to determine, whether there is a package-component mismatch. However, if after a scan the administrator is required to close the vulnerability and updates the system, and the vulnerability is still present, then there will be a more in-depth investigation. Therefore, penetration testing cannot immediately detect package-component mismatches but provides an opportunity to solve the problem.

7 Countermeasures

In this section, we propose the schemes that detect and fix the blind spot issue. The proposed schemes can deal with the blind spot type II.

7.1 Package-Component Mappings

To ensure package-component consistency, knowledge of which component configuration is appropriate for the current update state is crucial. Fig. 6 shows the registry paths for package-component mappings, where it can be determined which version of a particular component belongs to which update package. The subkeys of "ComponentDetect" represent component names. The red box to the right lists the version of the component installed by the particular package. For example, the name of the package that installed version 6.1.7200.21980 of the amd64_microsoft-windows-os-kernel is KB3046480.



```
App Management
App Paths
Applets
Audio
Authentication
BitLocker
BITS
Component Based Servicing
  CatalogsToUninstall
  ComponentDetect
    amd64_microsoft-windows-a..ence-mitigations-c1_31bf3856ad364e35_0.0.0.0_r
    amd64_microsoft-windows-a..ence-mitigations-c3_31bf3856ad364e35_0.0.0.0_r
    amd64_microsoft-windows-activexcompat_31bf3856ad364e35_0.0.0.0_none_764
    amd64_microsoft-windows-appid_31bf3856ad364e35_0.0.0.0_none_d744a17cb2
    amd64_microsoft-windows-csrsrv_31bf3856ad364e35_0.0.0.0_none_497fc836a5e
    amd64_microsoft-windows-eosnotify_31bf3856ad364e35_0.0.0.0_none_659cc4f
    amd64_microsoft-windows-os-kernel_31bf3856ad364e35_0.0.0.0_none_ec28f2d1b4a bfd4
    amd64_microsoft-windows-r..gistry-trustedtypes_31bf3856ad364e35_0.0.0.0_no
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Package_2_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.Trigger_1	REG_SZ	6.1.7601.21701@@2
Package_2_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.Trigger_1	REG_SZ	6.1.7600.21980@@2
Package_3_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-6_neutral_LDR	REG_SZ	6.1.7601.21701@2
Package_3_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-7_neutral_LDR	REG_SZ	6.1.7601.21701@4
Package_3_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-8_neutral_GDR	REG_SZ	6.1.7601.17592@2
Package_3_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.Trigger_1	REG_SZ	6.1.7600.21980@@2
Package_4_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.3046480-10_neutral_LDR	REG_SZ	6.1.7600.21980@4
Package_4_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.3046480-11_neutral_GDR	REG_SZ	6.1.7600.17795@2
Package_4_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.3046480-9_neutral_LDR	REG_SZ	6.1.7600.21980@2
Package_6_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.Trigger_1	REG_SZ	6.1.7600.20941@@2
Package_7_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-16_neutral_LDR	REG_SZ	6.1.7600.20941@2
Package_7_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-17_neutral_LDR	REG_SZ	6.1.7600.20941@4
Package_7_for_KB2534366~31bf3856ad364e35~amd64~~6.1.1.0.2534366-18_neutral_GDR	REG_SZ	6.1.7600.16792@2
ckage_7_for_KB3046480~31bf3856ad364e35~amd64~~6.1.1.2.Trigger_1	REG_SZ	6.1.7600.21980@@2

Fig. 6. Registry paths for package-component mappings

We can refer to the file information for updates provided by Microsoft. Since October 2016, Windows supports the cumulative update method [32]. Microsoft began to provide file information with the monthly cumulative update. The file information for a cumulative update is in CSV format. It contains the information about files to be updated, versions, and release dates. By collecting that information, we can extract the package-component mapping list.

7.2 Detecting Package-Component Mismatches

As discussed in Section 7.1, the package-component mappings are the key information to solve the blind spot issue. The detection scheme requires the following information described in Table 8. The following procedure describes the detection scheme.

1. *List the installed packages on the system*
   The package installation information can be found in the registry path P_n. Alternatively, if you run the command "dism /online /get-packages," the servicing agent queries the registry keys and values under the path P_n.
2. *Check the package-component mappings*
   It must be ensured that your system has component configurations that match the installed package. The mapping list introduced in Table 8. provides the answer. If all component configurations match the package-component mapping list, the update status is normal; unmatched items should be considered to have a package-component mismatch.
3. *Verify the hardlink information of components*

Finally, the hardlink information for each component needs to be verified. The "fsutil hardlink list [component file path]" command tells us the hardlink information. If the hardlink sources in the component store have the correct versions of the target components, the update status is normal. Otherwise, the update status has been compromised.

Table 8. Preparations for a detection scheme.

Item	Description
List of package-component mapping	• Information about which package contain which components • A record in the list comprises a package name, a component name, and a component version name. (For example, {KB001, Component_A, 1.0})
Package information	• Information of the components that make up the system (including component names and component versions)
Component information	• Information about all update packages installed on the system (including package names)

We assume that the correct package-component mapping list is extracted before the execution of Step 2. All the steps can be executed with user privileges. After the execution of Step 3, the compromised components will be identified.

7.3 Fixing Package-Component Mismatches

The fixing scheme is conceptually similar to that for update tampering method 1. The following procedure describes the fixing scheme.

1. *Detect the illegally replaced components*
   The illegally replaced components can be identified in the detection scheme.
2. *Correct the component versions in the registry*
   Change the registry value of C_v to the correct one
3. *Run the "SFC /SCANNOW" command*
   Let SFC replace the component files with the correct ones. After the execution of SFC, the system will require a reboot. Through the pending file rename operations after the reboot event, the mismatched components will be restored.

Because TrustedInstaller permissions are required to modify the component version names, Step 2 and 3 should be executed with Administrator privileges.

8 Discussion

In this section, we summarize features of the update state tampering technique, things defenders should take, and limitations of this work.

Attack Feature. The characteristics of the update state tampering technique are shown in Table 9. Adversaries may tamper with the Windows registry and the pending.xml file to generate known but invisible vulnerabilities on victim's system, or as part of other techniques to aid in persistence and execution. The technique affects all versions of Vista and later. An adversary must already be in a privileged user context (i.e., Administrator), The result of the technique bypasses the code integrity policy, SFC, Windows Update Check, and security compliance tools. An adversary can use this technique to revive remote code execution vulnerability as described in Section 4.

Table 9. Characteristics of update state tampering technique.

Update State Tampering technique	
Tactic	Persistence, Defense Evasion
Platform	Windows
Permissions Required	Administrator, TrustedInstaller
Data Sources	Windows Registry (C_v), pending.xml file (T_f, T_v)
Defense Bypassed	Code Integrity Policy, SFC, Windows Update Check, Security Compliance tools

Defender Considerations. Systems that have package-component mismatches are at heightened risk because of invisible security holes. A user can self-check file versions of components that have major vulnerabilities such as Eternal Blue [33]. And users can utilize the update state check scheme as described in Section 7 for batch verification of component versions. Those schemes can be easily created in the form of a Powershell script. We recommend that users periodically check the consistency between update packages and components installed in the system.

Limitations. The identified issues in Section 3 are not security vulnerabilities, rather than structural problems of the current Windows update management. Therefore, it is recommended to use the identified methods together with known exploits or 0-days. 0-day attacks are one shot only. Once patched, holes are closed. However, the proposed methods can widen the attack choices. Because of the blind spots, the tampered update state cannot be detected or fixed. The exposed vulnerabilities will persist until the next update to the replaced components.

As discussed in Section 4, it is most advantageous for an attacker to replace it with an older version of the component that has remote code execution vulnerability. However, there are a few cases where the update state tampering technique does not apply. If only individual components are replaced, the system may not work properly owing to inconsistencies with other components patched together. Therefore, an attacker should ensure that a system is running reliably after generating a package-component mismatch. We address this issue in the Component Dependency Problem subsection below. And the update state tampering technique assumes that there is a vulnerable version of the component inside the component store. This technique cannot be applied if the user has recently cleaned up the component store or if the operating system installed on the system is the latest version that does not require additional update installations. We address this issue in the Applicability of Attack Scenario subsection below.

The update state check scheme proposed in Section7 collects the component resource information and identifies the differences from the correct configuration values. The scheme does not consider rootkit attacks in this study. For example, if an attacker installs a malicious filter driver on a target system that already has control, the attacker could manipulate the target to always return the correct value when checking the component resource information. In this case, the update state check scheme is disabled.

Also, the proposed scheme deals with type II blind spot. However, it does not handle type I blind spots. To remove type l blind spots, a filtering module in the form of a kernel driver should be deployed. However, filtering of loaded components is a sensitive issue. A careless design can cause a new security module to be bypassed or cause system performance degradation. Eliminating type I blind spots will be focused on in future work.

Component Dependency Problem. In this study, we have not yet found an explicit method or theoretical model to identify component dependency problems. Security patches for components are implementation dependent. For example, in a security update, some code has weaknesses that correspond to CWE-120 [34], so the developer may only need to add logic to check the length of the input values without adding or removing features. In this case, you can expect the system to function normally even if the attacker replaces the component with a previous version of the component. However, not all updates will be the same as the example, and we cannot guarantee that the system will work. Also, Microsoft does not officially provide information that can identify component dependencies. Therefore, we recommend that attackers perform the test in the same environment as the target system before performing the attack. For testing purposes, the attacker should collect the base version name of OS, the list of installed packages, and the presence of previous version components on the target system. We will cover a virtual machine-based automated testing system that can identify component dependencies in future work.

Applicability of Attack Scenario. In SP 800-61 [35], NIST introduces several example actions in the recovery phase of the incident response. Table 10 shows the applicability of the proposed attack scenario when each action in the recovery phase is taken. We assume the case of ④ in the attack scenario. The attack scenario is not affected by the actions of ⑤ and ⑥. The actions of ① and ② involve re-imaging the machine. In such cases, the proposed scenario is of no use. And the attack scenario is considering the action ③ but assumes that the tampered components are left undetected due to the blind spots.

Table 10. Applicability of the proposed attack scenario due to actions at a recovery phase

Action	Applicability	Action	Applicability
① Restoring system from clean backups	X	④ Installing patches	O
② Rebuilding systems from scratches	X	⑤ Changing passwords	O
③ Replacing compromised files with clean versions	△	⑥ Tightening network perimeter security	O

Real World Survey. To the best of our knowledge, the Windows update management problems identified in this paper have not yet been reported. However, there is a possibility that this issue will be exploited without being disclosed. We have not done enough research in this study to see if the proposed techniques are being utilized in the real world. We plan to deploy the detection script for the proposed technique, collect the actual cases, and then evaluate the effectiveness and practical impact of the proposed technique and the attack scenario. The Powershell script has been publicly uploaded to GitHub.[1]

9 Conclusion

In this paper, we present Update State Tampering, a novel post-compromise technique exploiting the package-component mismatch and the blind spot issue. Following the update state tampering technique, the system loads the old versions of components that contain vulnerabilities. Further, the Update Check, the System File Check, and the Security Compliance tools will not detect any abnormalities. These issues affect all versions of Vista and later. To rectify these problems, we proposed update state check scheme, which detects and corrects package-component mismatches.

Acknowledgement. We would like to thank our shepherd Adam Doupe and our anonymous reviewers for their valuable comments and suggestions. We would also like to thank Sungryoul Lee, Seunghun Han, Junghwan Kang, Hyunyi Yi and Wook Shin for their feedback and advice.

References

1. B. E. Strom, J. A. Battaglia, M. S. Kemmerer, W. Kupersanin, D. P. Miller, C. Wampler, S. M. Whitley, and R. D. Wolf. Finding Cyber Threats with ATT&CK^{TM}-Based Analytics, MITRE TECHNICAL REPORT, 2017.
2. The MITRE Corporation. Presentation: Detecting the Adversary Post-Compromise with Threat Models and Behavioral Analytic, Available at: https://www.mitre.org/publications/technical-papers/presentation-detecting-the-adversary-post-compromise-with-threat, Accessed February 27, 2018.
3. S. Yadav and D.Mallari. Technical aspects of cyber kill chain. 3rd Communications in Computer and Information Science, vol. 536, pp.438–452, 2016
4. P. Chen, L. Desmet, and C. Huygens. A Study on Advanced Persistent Threats. In Proceedings of the 15th IFIP TC6/TC11 Conference on Communications and Multimedia Security, 2014.
5. S. Malone. Using an Expanded Cyber Kill Chain Model to Increase Attack Resiliency. Black Hat US, 2016
6. V. Smith, and C. Ames. Meta-Post Exploitation, Black Hat US, 2008
7. The MITRE Corporation. ATT&CK Matrix, Available at: https://attack.mitre.org/wiki/ATT&CK_Matrix, Accessed February 27, 2018.
8. P. Speulstra. Accessibility Features, https://attack.mitre.org/wiki/Technique/T1015

[1] Update State Checker Project GitHub: https://github.com/ksj1230/Update-State-Checker

9. C. Tilbury. Registry Analysis with CrowdResponse, Availble at: https://www.crowdstrike.com/blog/ registry-analysis-with-crowdresponse/, Accessed February 27, 2018.
10. B. Jerzman, and T. Smit. Modify Registry, https://attack.mitre.org/wiki/Technique/T1112
11. Kaspersky Lab. The Regin Platform Nation-State Ownage of GSM Networks, Available at: https://securelist.com/files/2014/11/Kaspersky_Lab_whitepaper_Regin_platform_eng.pdf, Accessed February 27, 2018.
12. FireEye Threat Intelligence. APT28: A Window Into Russia's Cyber Espionage Operations? Available at: https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-apt28.pdf, Accessed February 27, 2018.
13. R. Falcone. Shamoon 2: Return of the Disttrack Wiper, Available at: https://researchcenter.paloaltonetworks.com/2016/11/unit42-shamoon-2-return-disttrack-wiper, Accessed February 27, 2018.
14. Microsoft. Use the System File Checker tool to repair missing or corrupted system files, Available at: https://support.microsoft.com/eu-es/help/929833/use-the-system-file-checker -tool-to-repair-missing-or -corrupted-system-files, Accessed February 27, 2018.
15. Microsoft. How to get an update through Windows Update, Available at: https://support.microsoft.com/en-us/help/3067639/how-to-get-an-update-through-windows-update, Accessed February 27, 2018.
16. Microsoft. Microsoft Baseline Security Analyzer, Available at: https://technet.microsoft.com/en-us/security/cc184924.aspx, Accessed February 27, 2018.
17. Microsoft. Understanding Component-Based Servicing, Available at: https://blogs.technet.microsoft.com/askperf/2008/04/23/understanding-component-based-servicing/, Accessed February 27, 2018.
18. Microsoft. Manage the Component Store, https://technet.microsoft.com/en-us/library/dn251569.aspx, Accessed February 27, 2018.
19. M. E. Russinovich, D. A. Solomon, and A. Ionescu. Windows Internals 6th part2, pp. 525
20. Microsoft. Code Integrity, https://technet.microsoft.com/en-us/library/dd348642(v=ws.10).aspx, Accessed February 27, 2018.
21. Microsoft. Kernel-Mode Code Signing Walkthrough, Available at: https://msdn.microsoft.com/en-us/library/windows/hardware/dn653569(v=vs.85).aspx, Accessed February 27, 2018.
22. The MITRE Corporation. CVE-2017-0114, Available at: https://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144, Accessed February 27, 2018.
23. S. Frei, M. May, U. Fiedler, and B. Plattner. Large-scale vulnerability analysis. SIGCOM M workshop on LSAD, 2006
24. H. Joh, Y. K. Malaiya. Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics. International conference on Security and Management (SAM), 2011
25. Microsoft. Process Monitor v3.50, Available at: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon, Accessed February 27, 2018.
26. AhnLab. MyPCInspector, Available at: http://www.ahnlab.com/kr/site/product/productView.do?prodSeq=86, Accessed February 27, 2018.
27. Rapid7. Metasploit, Available at: https://www.metasploit.com/, Accessed February 27, 2018.
28. OpenVAS. OpenVAS, Available at: http://www.openvas.org/, Accessed April 20, 2018.
29. Greenbone Networks. Greenbone, Available at: https://www.greenbone.net/en/, Accessed April 20, 2018.
30. Tenable. Nessus Home, Available at: https://www.tenable.com/products/nessus/nessus-professional, Accessed April 20, 2018.

31. Rapid7. Nexpose, Available at: https://www.rapid7.com/products/nexpose/, Accessed April 20, 2018.
32. Microsoft. Further simplifying servicing models for Windows 7 and Windows 8.1, Available at: https://blogs.technet.microsoft.com/windowsitpro/2016/08/15/further-simplifying-servicing-model-for-win dows-7-and-windows-8-1/, Accessed April 20, 2018.
33. Microsoft. How to verify that MS17-010 is installed, Available at: https://support. microsoft.com/en-us/help/4023262/how-to-verify-that-ms17-010-is-installed, Accessed February 27, 2018.
34. The MITRE Corporation. CWE-120, Available at: https://cwe.mitre.org/data/definitions/ 120.html, Accessed April 20, 2018.
35. P. Cichonski, T. Millar, T. Grance, and K. Scarfone. Computer Security Incident Handling Guide. NIST Special Publication, 800-61, 2012