Windows Installer Security

C. Kadluba[1], M. Mulazzani[2], L. Zechner[2], S. Neuner[2], E. Weippl[2]
[1]University of Applied Sciences Technikum Wien
[2]SBA Research
christian.kadluba@gmail.com, {mmulazzani, lzechner, sneuner, eweippl}@sba-research.org

Abstract

Windows Installer has been an integral part of Microsoft Windows for a long time and is the standard method of software management and deployment on this operating system. Since this technology exists for quite a while and is heavily used, it is worth to take a look at it from the security perspective, in particular regarding the risks users face. This paper shines light on the risks that can arise from the content in untrusted installer packages. For one, we analyze the variable importance of different data regions within an msi file and the consequences of flipping a single bit, possibly resulting in corrupted content and/or installation logic. A specially developed analysis script shows that malware detection in MSI files can be significantly improved compared to normal scans with conventional anti-virus products. The method is tested with MSI packages prepared with malware samples and the results are compared to normal AV scanning. Lastly, we created a metric to allow the advanced users to evaluate the possible risks of a given msi installer file prior to installation. Installer packages of freely available, and very popular products are analyzed with this scripts to get a picture of the current practice of authoring security during setup throughout the software industry.

1 Introduction

Windows Installer is an integral part of the Microsoft Windows operating system and as such responsible for installing, uninstalling and changing applications on a computer. In the world of Windows it is the standard method of software deployment. Typically, the software to be installed is packaged together with the setup logic in a file that has the extension *.msi*, which is called an installer package. Windows Installer was first released 1999 together with Microsoft Office 2000, and has been installed on every Windows machine since. During installation of msi files the contents of the msi package are extracted by the Windows Installer and the system is modified as specified by the author of the package. Some of those modifications require more privileges than normally granted to the desktop user, therefore parts of Windows Installer run with system privileges as described in [6].

This paper focuses on the risks and threats which can arise from installing and executing untrusted msi packages. When users download packages from the Internet, it is hard to find out what the package will actually do on the system. This includes the types of actions the package will perform when being installed as well as the privilege level that it will request. The second topic is malware detection in msi packages, which we will show is poor compared to standard signature-file based malware detection as deployed on client computers today.

This paper shows two new methods of statically analyzing msi packages without installing them. The first method shows how to improve the detection rate of malware scanners in msi files and also gives information about criticality depending on execution privileges and other information extracted from the msi file. The method is evaluated by scanning infected msi files and comparing the results to scans of the statically analyzed files using conventional AV scanners. This advanced analysis technique incorporates the extracted files as well as the metadata from msi files and analyzes them with any installed AV product. The second part demonstrates a method to analyze an msi file and detect authoring that is not malicious but still could cause unintended risks for the user. Such authoring includes excessive use of Custom Actions and elevated privileges. A script was developed that does a number of checks on an msi file and calculates a risk score based on the results. A selection of popular msi files, like the installers of Google Chrome or Skype, is analyzed with the script and the results are discussed. This gives an impression about the msi security awareness of software vendors.

To summarize, the contributions of this papers are:

- We analyze the internals of msi files using file fuzzing.
- We analyze malware and AV scanners with respect to msi files.
- We present a risk metric based on our insights into msi files.
- We will release the code under an open source license[1].

The rest of the paper is organized as follows: Section 2 gives a brief background on msi files and related work. Section 3 describes our fuzzer and how it was used to analyze

[1]`https://github.com/kadluba/MsiScripts`

different msi files by looking at the resulting error codes. In Section 4 we analyze malware in msi files, and evaluate it using virustotal.com as well as Microsoft Security Essentials. In Section 5 we present our risk metric for msi files and evaluate it with popular real-world msi installer files. We discuss our results in Section 6, before we conclude in Section 7.

2 Background

Windows Installer packages are the de facto standard for installing software in Windows and as such they usually contain executable code. However, the format of msi is not officially specified and also not covered by Mircosoft's Open Specification Promise[2]. But, for one, there is the executable of the application to be installed, which is copied to the system during installation. On the other hand the packages can contain Windows Installer Custom Actions. These are typically program files embedded in the package which are used to carry out installation tasks that cannot be accomplished by the built-in Windows Installer functionality and are specified by the creator of the msi file. Msi files can furthermore be packed or fragmented since msi packages are essentially a file system within a file [13]. Additionally the msi installer packages contain a structure of features and components which are combined with conditions and execution privileges. This can be understood as the setup logic of the package and thereby controls the installation of program files and execution of Custom Action binaries.

Beside the program files, an msi file also contains a relational database consisting of a set of standardized tables as described in [9]. This Installer Database contains all information necessary to install the product including the aforementioned setup logic. The Binary table also contains binary files used during the installation, which can include DLLs executed by Custom Actions. Most of the setup logic in the packages is in fact a declarative description of how the installed application should look like. How the desired system state is achieved is mostly up to the routines implemented in the Windows Installer service. The concept is to describe the installation rather than to script it. Custom Actions allow the authors of msi packages to execute custom code during the installation. This code can be either included in the package, mostly as embedded DLLs, or it could come from another source, like an executable which is already present on the system. With Custom Actions, Windows Installer essentially allows execution of arbitrary code with system privileges.

2.1 Related Work

There is not much work available in the literature on the security of msi files or packet managers in general. Cappos et al. [1] analyzed the security of multiple packet managers like APT and YUM for Linux and Unix, whereas [15] discusses improvements to the security of these packet management systems regarding authenticity and integrity. Di Ruscio [14] describes strategies to overcome problems between upgrades. In the area of static analysis for software installation, recnet work focused on analyzing Android APKs since the Android ecosystem has a special permission set for applications [2] and APK files can be leveraged for static analysis [5, 16]. Recent work in the area of malware detection focuses on the usage of packers [11, 12, 17], which is related to our work with regard to msi files since msi files (as we will show later) can be used to reduce detectability of contained malware for standard anti-virus detection.

3 Fuzzing MSI Packages

To assess to what extend msi packages are prone to manipulation and its possible consequences we wrote a simple fuzzer to manipulate msi packages and tested the consequences by collecting the error codes of Windows Installer.

3.1 msi Fuzzer

Our fuzzer iterates over a given msi package and changes bits to measure the consequences on the installation and deinstallation process. A powershell script automates this, and the bits chosen to be manipulated are either chosen at random or sequentially. Depending on the size of the msi package, sequentially flipping all bits can take a considerable amount of time since the software will be installed and removed again. If you consider a msi package with 100 kilobytes and assume that installation and removal takes approximately 10 seconds, this results in a total runtime of 95 days. Logging mechanisms keep track of the particular bits flipped via offset, the date and time, and the return value of msciex after trying to install it.

3.2 Conducted Tests

We initially implemented the random selection of bits to flip, in order to get a fast estimation of results and the corresponding error codes and consequences. For evaluation we fuzzed two packages in sequential mode: for one WixLanguage.msi, which is a simple 124 kilobyte example msi from the book [6] without Custom Actions. The other package was a simple msi with an Authicode signature named SignedSetup.msi. It was created using Windows Installer XML and signed using a self-signed certificate from the book. To speed up the process we split the fuzzing across multiple virtual machines.

The different error codes we observed were as follows: 0, 1603, 1605, 1613, 1620, 1623, 1625, 1633.

- 0 denoted that the installation was successful.
- 1603: fatal errors during installation
- 1605: action only allowed for installed packages
- 1613: A higher version of Windows than the one currently running is needed.

[2]`http://www.microsoft.com/openspecifications/en/us/\programs/osp/default.aspx`

- 1620: unable to open installation package
- 1623: package language not supported
- 1625: installation prevented by system policy
- 1633: unable to install on this platform

Error -100 in the Figure below is special so far as it is not an actual error code from Windows Installer, but an arbitrary value assigned from us for packages which we could install, but did not remove afterwards. The second package had the additional error codes 87 (parameter is invalid), 1601 (unable to access Windows Installer) and 1631 (unable to start Windows Installer daemon). The results are visualized in Figure 1 for WixLanguage.msi and Figure 2 for SignedSetup.msi. Please note that the x-axis represents the offset in the file.

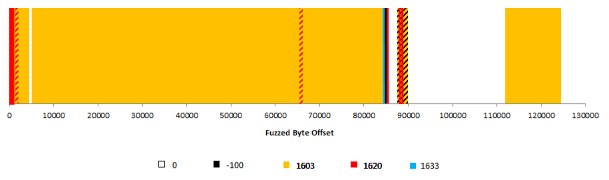

Figure 1: Error codes when fuzzing WixLanguage.msi

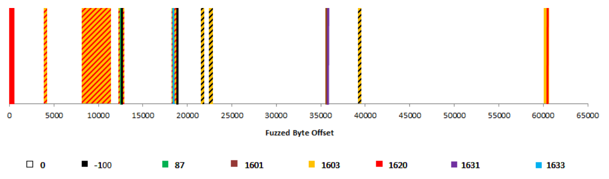

Figure 2: Error codes when fuzzing SignedSetup.msi

3.3 Discussion of Results

On a first sight there was not much similarity between the results from fuzzing those two packages. However, the first 512 bytes had almost identical content and error codes, as they contain the header information for the entire package. Furthermore, both packages had areas filled with zeroes where there was no error induced due to fuzzing. We speculate that these areas are not used during the installation process. We also found that the signature of the second msi file had no consequence on the ability to install the fuzzed msi. Apparently it is only used to assess whether the issuer is trusted, but not for protecting the integrity of the files.

4 Malware in MSI

The existence of executable code within the msi files makes them a subject for anti-virus scanning. The binaries included in the packages could be infected with malware, so scanning for viruses is desirable before installing them. This is referred to as "baseline ecosystem cleaning" as explained in [18]. We present a new method to improve the effectiveness of conventional anti-virus products when scanning msi files and compared to conventional scanning of the msi files.

4.1 msi Files Extractor

To analyze executable code from the different possible sources within an msi file, we wrote a script that extracts the different types of content of a given msi file. We then scanned the extracted files with an installed AV scanner. The setup logic of the msi file is taken into account so that a better risk assessment is possible when infected binaries are found. Our script also analyzes parts of the msi file which are never executed or used during setup. While this might be overprotective, it is worth questioning why a possibly malicious file is shipped with an msi file, but not used during installation. Every infected file should be taken seriously since the actual infection and execution of malicious code could take place by other means than the Windows installer and can therefore not be detected exclusively by looking at the setup logic and structure within the msi file. However, knowing about the installation behavior of infected contents of an msi file can serve as an additional guideline for risk assessment. We then use the extracted files to assess whether the extraction of files increase the likelihood for anti-virus products to detect them. Since content in msi files can be packed and fragmented [11, 12] the detection rate is expected to increase as anti-virus engines are in general not expected to extract this custom file format logic during signature-based analysis.

In the first evaluation step the msi script extracts all files and tables from the msi package. The tables will be extracted as simple text files with tabulator-separated values and header lines with the column names and data types. Those files will then be parsed by the further analysis steps of the script. The script then iterates over the extracted program files. Each of them is scanned with the installed AV scanner which is not specified by the script and treated as a black box. If an infection is found, the extracted tables File, Component and FeatureComponent are analyzed to find out if the file can actually be installed or if it's only an orphaned part of the msi package belonging to no feature. Another possibility is that the file is part of one or more features but its installation is disabled by a logical condition which is always false. Both cases are common if setup developers disable a part of the product without actually removing it from the msi package. This approach is equivalent to commenting some lines in the source code of a program instead of deleting them. In this way the removed parts can be easily restored later, if the product management changes their mind. The script displays a message for every infected program file. If the file seems not to be installable according to the setup logic only an informational message is displayed, whereas a warning message appears for infected files which may actually be installed.

In the last step, the script iterates over the files extracted from the Binary table. Every binary file is scanned for known viruses by the local anti-virus engine. If an infection is found, the tables CustomAction, InstallUISequence, InstallExecuteSequence and ControlEvent are searched for the infected Binary table file. Like before, this step is done to find out if the file is actually used by any Custom Action and if logical conditions do not prohibit its execution. Additionally it is checked if the Custom Action can run with user privileges or with system privileges (elevated). Like for the program files in the previous step, the message shown by the script for infected Binary table files depends on the actual usage of the file and the privileges it is allowed to run with.

4.2 Conducted Tests

For evaluation of our msi extraction analysis, we generated a set of msi test files. Four base packages were created and used for analysis: *Package 1* contains infected program files which are installed during setup. *Package 2* contains infected program files which are not used during installation, thus representing the previously described possibility of malicious dead code. *Package 3* contains infected Binary table items which are used in Custom Actions, whereas *Package 4* contains infected Binary table items which are not used for any Custom Actions. We created mutations of those four baseline packages, including various malware samples. The malware samples were downloaded from `http://www.eicar.com` and the Open Malware Database at `http://www.offensivecomputing.net`. Table 1 shows a list of the used malware samples. To create the different msi installer files we used the Windows Installer Xml Toolset 3.6.3303.1.

Malware	**Description**
Eicar	EICAR-AV-Test
Parite	W32/Parite.B
Hahor	VirTool.Hahor.A
Lolol	Worm/Lolol
Webdav	Exploit.Webdav-4
Iexploiter	Backdoor.Iexploiter.A
Bifrose	Backdoor.Win32.Bifrose.d
Setcrack	Trojan.Win32.Setcrack

Table 1: Used Malware Samples

We combined the four base package types with all of the used malware samples, resulting in 36 msi files. For reference, we also scanned the initial malware code. The results during our evaluation are shown in Table 2. The number in the cells show the the number of scan engines that detect the malware using `http://www.virustotal.com`. This web service uses 46 different AV scanners to analyze an uploaded file regarding viruses. The number in each cell shows how many out of the 46 scanners detected an infection. The last column shows scan results when uploading the malware sample binary itself. This was done to verify if the sample could be detected by the used AV scanners at all and, more important to see if the scanners perform worse when the malware is inside an msi file compared to scanning the bare malware file alone.

	Package 1	**Package 2**	**Package 3**	**Package 4**	**Malware**
None	0	-	0	-	-
Eicar	16	16	14	14	42
Parite	17	17	23	22	42
Hahor	17	17	21	22	38
Lolol	18	18	22	22	42
Webdav	12	12	18	19	38
Iexploiter	15	15	21	21	39
Bifrose	17	17	23	23	39
Setcrack	17	17	23	23	40

Table 2: VirusTotal Scan Results

Table 3 and 4 show the results when the test packages were scanned using a local anti-virus scanner. In this particular experiment the locally installed scanner were Microsoft Security Essentials version 4.1.522.0 with virus definitions as of 2012-12-31 and AVG 2013.0.2805. Both are also part of the 46 scan engines used by VirusTotal and as such any other scan engine could have been used. This experiment was run to show the same methodology in local context. A checkmark in the table indicates that the infected file was correctly identified, respectively that the installer package without any malware was not classified as malicious.

	Package 1	**Package 2**	**Package 3**	**Package 4**
None	✓	-	✓	-
Eicar	✓	✓	✓	✓
Parite	✓	✓	✓	✓
Hahor	✓	✓	✓	✓
Lolol	✓	✓	✓	✓
Webdav	✓	✓	✓	✓
Iexploiter	✗	✗	✗	✓
Bifrose	✓	✓	✓	✓
Setcrack	✓	✓	✓	✓

Table 3: Microsoft Security Essentials Scan Results

4.3 Discussion of the Results

Regarding the results, we were able to show in the first experiment that most AV scanners have problems detecting infected files within msi files. This is probably because the files can be packed and fragmented within the msi package, which obscures signatures that AV scanners are looking for. The results of the scan with VirusTotal in Table 2 clearly indicate this. The last column shows that the majority of the 46 AV scanners could detect the malware sample when it was uploaded directly. In comparison, the other columns show that not even half of the scanners could detect the same sample when it was packed into an msi file. Further-

	Package 1	Package 2	Package 3	Package 4
None	✓	-	✓	-
Eicar	✓	✓	✓	✓
Parite	✓	✓	✓	✓
Hahor	✗	✗	✗	✓
Lolol	✓	✓	✓	✓
Webdav	✗	✗	✗	✓
Iexploiter	✓	✓	✓	✓
Bifrose	✓	✓	✓	✓
Setcrack	✓	✓	✓	✓

Table 4: AVG Scan Results

more it comes to no surprise that the anti-virus scanners do not take the setup logic and execution privileges of the infected contents into account. Therefore the VirusTotal scanners show all infected contents as equally critical. In the case of the test packages which do not install or use the infected contents (package 2 and 4) this can be considered as false positives. However, as noted before, any infected file should be taken seriously, even if the malicious content does not seem to be used.

Regarding the local scans with Microsoft Security Essentials as well as AVG we were able to see that both ignore the setup logic which are part of the msi files, as shown in Tables 3 and 4. The test files are classified as malicious even though the malware resides in dead code. As such our results indicate that for a qualified risk assessment the incorporation of logic is important. One thing that is interesting is that the Ieploiter malware sample could not be detected by Microsoft Security Essentials when it was used in any of the four test msi files, except in Package 4. Scanning the original Iexploiter malware sample file directly with Microsoft Security Essentials confirmed that it could not be detected by the scanner. Taking a closer look at the files exported from the msi packages showed that they get marginally changed by the tool we used, MsiDB.exe. Thus, Microsoft Security Essentials detected not the originally embedded malware in the mutated file from Package 4.

5 Detailed MSI Package Analysis

To ease the decision for users whether an untrusted msi file can do any harm to the user's computer, we developed a metric for assigning a risk score to unknown msi files. As outlined before, Windows Installer is able to do arbitrary system modifications with elevated privileges depending on the contents of the package processed. Essentially it is like running an unknown program which is possibly executed with system privileges. Therefore it is important to find out if it is in fact using system privileges or not and to get an overview about what types of actions it will execute. Even if an msi file contains no malware, it can still pose risks caused by undocumented and unexpected behaviour, system modifications or collection of private data. Additional risk can come from bugs in the setup logic, malicious modification of msi files by a third party, or wrong usage of the installer package.

Our risk metric is implemented in a script which a user can run prior to installing a package by statically analyzing the msi package. It determines if the msi file uses elevated privileges, checks for Custom Action types and other things. Regarding the technical capabilities of the Windows Installer, we believe that the use of Custom Actions should be discouraged, and should only be used if the necessary installation tasks cannot be carried out by Windows Installer's standard functionality. The reason why Custom Actions should not be used is not only security but because they have implications on the transaction safety of the installation process as described in [9]. Like Custom Actions, there is also a number of other possibly problematic options that could be used by the author of an msi package. Our tool tries to detect those and provides these information to the user which can be used to assess the possible risks before installing a package. The used analysis technique is comparable to the work described in [5], where the permission settings of Android APK images are statically checked. In our experiments the script is used to analyze msi packages of some popular products such as Google Chrome or Skype. This is done to get an idea of the security standard that software vendors apply when authoring their Windows Installer packages and the amount of possible risk that users of common msi packages are typically exposed to according to our metric.

5.1 Design and Risk Metric

In the first step our analysis script extracts all contents from the msi file. It then does a number of checks on the extracted table files, to find actions and intents which could be and are therefore considered dangerous in our analysis. We conduct different checks according to three categories, described below. However, our analysis is testing for the presence respectively absence of certain features which are possible with Windows Installer and within the setup logic. We do not analyze the code executed, and further analysis of the code is necessary to know what the code actually does. [3], [16] and [7] describe similar methods for static code analysis. An overview of all statically conducted checks can be seen in Table 5. Based on the results of the checks, a risk score is calculated and displayed. The higher the score, the higher the possible risk caused by the msi file.

The risk metrik is simple in regard that the higher the value, the higher the potential harm the msi file can do to the computer. The overall checks who score one point can be seen in Table 5. Custom Actions that run code from within the package add two points, while arbitrary system commands add five points to the overall risk score. The number of Custom Actions is irrelevant, since one is already enough to manipulate the system in a malicious way. Elevated Custom Actions add twice the points.

Type	Check	Description
Global Checks	Signature	Checks for valid Authenticode signature
	Read-only	Checks if the package is read-only during installation
	Disable Elevation	Checks if elevation is disabled for the package
	Admin Image	Checks if this package is an administrative image
	Compressed Media	Checks for external storage of program files
	Per-User Installation	Checks the default installation mode of the package
System Privilege Custom Actions	Embedded DLL	Checks for Custom Actions executing an embedded DLL
	File Execution	Checks for executed files installed by the package
	Command	Checks for Custom Actions that execute a command.
	Property	Checks whether a Windows Installer property is set.
User Privilege Custom Actions	Embedded DLL	Checks for Custom Actions executing an embedded DLL
	File Execution	Checks for executed files installed by the package.
	Command	Checks for Custom Actions that execute a command.
	Property	Checks whether a Windows Installer property is set.
	ControlEvent	Checks for ControlEvent Custom Actions

Table 5: All Checks for our msi risk assessement

Regarding the global security of every msi package, we check for certain specific flags or features. For one we test if the msi package has a *signature* as described in [8]. This signature protects against file corruption, and is not specific to Windows Installer. It is the same signature for all executables and dynamic libraries in Windows, and can be added with the SignTool from Microsoft. We also check whether the package is marked *read-only* during installation. If this is not set the setup logic and the binary files can be modified by Custom Actions or by external processes using the Windows Installer API, allowing for self-modifying code during setup. The third check tests if *elevation* is globally disabled for the package, as Windows Vista and later only run Custom Actions elevated if elevation is not globally disabled for the package. We also test if the package is an *administrative image*. Administrative images are a Windows Installer feature often used by administrators to provide pre-configured installations, and they usually consist of the msi file accompanied by other files. Therefore they offer more possibilities for malicious manipulation compared to a single msi file. *Compressed Media* checks if the media cabinets which containing the program files to be installed are externally stored. This method has the same problem as administrative images, and protection against file manipulation is more difficult than for a single file. Lastly, *Per-User Installation* checks the default installation mode of the package, as Windows Installer offers two installation modes: Per-User and Per-Machine. Per-User installs are preferred as their capabilities are limited compared to Per-Machine installs.

The second category of checks is conducted with regards to Custom Actions which run with elevated privileges. An msi package creator can specify that those custom actions run in the context and with the privileges of the system account [10] depending on the elevation setting of the whole package and the configuration of the machine as documented in [9]. These Custom Actions are very critical as they can potentially make any changes on the system. Custom Actions can execute various vectors with system privileges: *embedded dynamic libraries (DLL)*, which are part of the msi package, *files* that are part of the msi package and *commands* that are defined within the Custom Action and executed with system privileges. These executed commands are considered as very critical, since they can execute any program the system has access to. This includes programs on removable media and network drives. It cannot be foreseen what programs will be accessible during the installation, therefore this type of Custom Action contributes the most to the risk score. The forth check tests for Custom Actions that set a Windows Installer *property*, which are variables that can be used by the setup logic during the installation.

Table 5 further shows checks for custom actions that run with user privileges. They are not as dangerous as elevated Custom Actions but still a possible risk since they can also be used to execute code. Like before with the elevated Custom Actions, we test for execution of embedded *DLLs*, *files* and *commands*, as well as if the package sets a *property* during execution. The only additional check is whether the msi contains *ControlEvent* Custom Actions. These are Custom Actions triggered by controls on the setup dialogs like buttons and can execute code embedded in the msi package similar to embedded DLLs. They always run with user privileges. Finally it should be mentioned again that we only check if Custom Actions of these various types exist and are used during the installation. Examining the code of the Custom Actions is not part of the process.

5.2 Conducted Tests

For evaluating our derived security risk metric, we used a small sample of publicly available msi files for standard software obtained from the internet. Table 6 lists the msi files which were used in the tests.

We then calculated the risk score and evaluated the previously described static checks as explained above. For better readability, the results are split in three different tables. Table 7 shows the results of the global checks. Table 8 then shows the results regarding the Custom Actions per package. The numbers in the cells indicate the number of Cus-

Product	**Version**	**Filename**
7zip	9.20 64 Bit	7z920-x64.msi
Apache Webserver	2.2.22	httpd-2.2.22-win32-x86-openssl-0.9.8t.msi
Autopsy Forensics Tool	3.0.1	autopsy-3.0.1.msi
Google Chrome	23.0.1271.97	GoogleChrome Standalone Enterprise.msi
LibreOffice	3.6.4	LibreOffice-LibO_3.6.4_Win _x86_install _multi.msi
MsiVal2 Tool	(MS Windows SDK)	MsiVal2.msi
Orca msi Editor	(MS Windows SDK)	Orca.msi
Skype	6.0.126	SkypeSetup.msi
TortoiseSVN SVN	1.7.9.23248	TortoiseSVN.msi

Table 6: msi Packages used for Risk Analysis

tom Actions found in the package with system privileges as well as user privileges. The calculated risk metric for each package is shown in the last column.

	Signature	**Read-only**	**Disable Elevation**	**Admin Image**	**Compressed Media**	**Per-User Installation**
7zip	✗	✗	✗	✗	✓	✓
Apache	✗	✗	✗	✗	✗	✓
Autopsy	✗	✓	✗	✗	✓	✓
Chrome	✓	✗	✗	✗	✓	✗
LibreOffice	✓	✓	✗	✗	✗	✗
MsiVal2	✗	✗	✗	✗	✓	✓
Orca	✗	✗	✗	✗	✓	✗
Skype	✓	✗	✗	✗	✓	✗
TortoiseSVN	✓	✗	✗	✗	✓	✗

Table 7: Results of Package Global Checks

5.3 Discussion of Results

Table 7 with the results of the package-wide checks shows that only four out of the nine analyzed packages contain a signature. This is already very prominent indicator for a lack of msi security awareness among the vendors. An impression that gets even stronger when noticing that almost none of the packages use write protection during the installation. Without that protection it is possible to manipulate the package during the installation process, for instance by Custom Actions. This can be used to implement non-obvious setup behavior, which cannot be easily analyzed in general. Another bad practice is that none of the packages disables elevation, not even those which obviously do not need it for any Custom Actions. If elevation is not needed it should be disabled globally. Otherwise it could be exploited by attackers who maliciously manipulate Custom Action code to gain system wide access. A positive observation is, that only two of the packages use external media cabinets. All others have the cabinets embedded in the msi file directly, as described in [4]. This is preferable because only that single msi file has to be protected against manipulation, for instance by using file system access control or checksums.

Table 8 with the results of the checks for elevated Custom Actions shows an alarming picture. Only the products MsiVal2, Orca and 7zip show a good example of msi authoring since they completely abandon Custom Actions that execute code. Like already mentioned, elevated Custom Actions can run with system privileges and are therefore able to do any possible modifications of the system. Custom Actions should only be used if there is absolutely no other way to perform the desired installation actions with Windows Installer standard methods. And if Custom Actions need to be used, then they should be running with user privileges. But the tested packages show a different picture. A very bad example is from our set the installer of Google Chrome. After deeper analysis of the package it appears that it does not use any of the standard Windows Installer functionality, like copying files onto the system. This is indicated by the fact that the package contains no file entries or any other information in the msi tables. There is only one big elevated Custom Action which seems to execute a proprietary installer. In this case Windows Installer is only used as a vehicle to get the proprietary installer to run with system privileges. This is considered bad practice, as it is unknown what that elevated Custom Action will do and should be considered a risk from the security point-of-view. User privilege Custom Actions are considered to be less dangerous than their elevated pendants, and can be seen to be by far less popular.

6 Discussion

Our file fuzzer showed that during installation the signature is only tested against a list of trusted software issuers, but there is no verification of the integrity of the package. This is troublesome since an adversary who is able to modify a msi may be able to piggyback an additional malicious payload. The majority of errors for the two msi files fuzzed were 1603 and 1620, meaning a fatal error occured during installation and that the package could not be opened. Most interestingly, we were able to create packages that could be installed without error, but

	Embedded DLL	File Execution	Command	Property	Embedded DLL	File Execution	Command	Property	ControlEvent	Risk Score
	System Privilege				User Privilege					
7zip	-	-	-	-	-	-	-	-	-	**9**
Apache	2	6	-	3	1	-	-	-	1	**19**
Autopsy	4	-	-	5	-	-	-	5	2	**10**
Chrome	2	-	1	10	-	-	-	-	-	**23**
LibreOffice	26	-	-	10	-	-	-	3	7	**12**
MsiVal2	-	-	-	1	-	-	-	-	-	**9**
Orca	-	-	-	4	-	-	-	-	-	**10**
Skype	17	-	-	21	-	-	-	-	2	**11**
TortoiseSVN	1	-	-	6	-	-	-	1	2	**11**

Table 8: Results of Custom Action Checks and Risk Score

not removed afterwards. This is especially dangerous for environments where software is automatically installed on a large number of clients, since the removal of potentially malicious software has to be done by hand.

Overall, the results of our experiments showed that developers as well as users and anti-virus solutions are not fully aware of the possibilities and risks of msi files. We showed in the first experiment that most conventional AV products fail to detect malware when it is packed inside of an msi file, whereas the same products could properly detect the virus when the malware binary itself is scanned. This strongly indicates that the initial hypothesis is correct, i.e. that AV scanners have a poor detection rate on files inside of msi files. That poor detection may come from the packing and fragmenting that is applied to the malware when they are stored inside the msi package. Most scanners do not seem to unpack the package contents and therefore often fail to detect contained malware. Another aspect of this experiment was to get better information about how critical an infected file inside an msi might be and adding contextual information of the msi file during analysis by considering the actual usage of the infected content and execution privileges. It would be wrong to speak about avoiding false positives here. Even infected content, which is not used by the msi package in an obvious way, could be triggered in a hidden fashion. Therefore every infection must be taken seriously. Still, additional information as provided by the script can serve as a starting point for further examination of infected msi files. The experiment showed that it is possible to gain such information by using our analysis methodology and implementation as described. The accuracy of the results from the evaluation was very high and the gained information very detailed.

The last experiment showed that it is possible to gain information about the potential risk arising from msi packages by statically analyzing them. The analysis tries to find the maximum impact an msi package can have on a system by looking at used Custom Actions, privileges and other characteristics. A test with some popular real-world msi packages showed that software vendors typically do not pay enough attention to their installers especially when it comes to clean and defensive authoring of the msi packages. Many Custom Actions and usage of unnecessary privileges combined with abandonment of digital signatures as documented in [8] make msi packages an ideal vehicle for malware. Companies will have to secure their software library consisting of msi packages on a file server by external means because the packages themselves are mostly insecure by design. Such packages, which are then automatically deployed to thousands of clients and servers within a company, must be an appealing target for malware authors and other malicious parties. When putting all those findings together, a concerning picture of msi security is drawn. Related papers [1, 15, 14] have shown that installer systems are security-critical and that vendors should put more effort into creating secure packages.

6.1 Future Work

For future work we plan to increase the sample size and evaluate our risk metric with a large set of freely available msi files. Even though our sample can be at the most considered as only exemplarily, many software vendors distribute their software as msi files. Furthermore we would like to deeply analyze available malware which is distributed or abusing msi files in the wild.

7 Conclusion

Within this paper we analyzed the security aspects of msi packages for Windows Installer with respect to malware detection as well as their defensive behaviour. We showed that while Windows Installer itself offers important baseline security functions and mechanisms, they are not commonly used for software which is available online. This is also true for very large software companies. On the other hand, most anti-virus scanners are not ready for msi, as they often fail to detect malware embedded in msi files, while also ignoring corner cases where the malware is not executed during the installation of the package. Our results indicate that software packaged in msi files can be an attractive target of and infection vector for malicious software.

Acknowledgements

This work was funded by COMET K1, FFG - Austrian Research Promotion Agency.

References

[1] J. Cappos, J. Samuel, S. Baker, and J. Hartman. A look in the mirror: Attacks on package managers. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 565–574. ACM, 2008.

[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[3] W. Fu, J. Pang, R. Zhao, Y. Zhang, and B. Wei. Static detection of api-calling behavior from malicious binary executables. In *2008 International Conference on Computer and Electrical Engineering*, pages 388–392. IEEE, 2008.

[4] T. Jaffri and K. Karnawat. Efficient delivery of software updates using advanced compression techniques. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM'06)*. IEEE, 2006.

[5] R. Johnson, Z. Wang, and C. G. A. Stavrou. Analysis of android applications permissions. In *2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, pages 45–46. IEEE, 2012.

[6] A. Kerl. *Inside Windows Installer 4.5*. Microsoft Press, 2008.

[7] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 329–240. ACM, 2012.

[8] Microsoft. *Microsoft Developer Network – Introduction to Code Signing*. `http://msdn.microsoft.com/en-us/library/ie/ms537361.aspx`, retrieved 2013-01-06.

[9] Microsoft. *Microsoft Developer Network – Windows Installer Documentation*. `http://msdn.microsoft.com/library/cc185688(VS.85).aspx`, retrieved 2012-07-22.

[10] Microsoft. *Microsoft Knowledgebase – How the System account is used in Windows*. `http://support.microsoft.com/kb/120929`, retrieved 2013-01-05.

[11] S. Mody, I. Muttik, and P. Ferrie. Standards and policies on packer use. In *Virus Bulletin Conference September 2010*, pages 372–280. Virus Bulletin Ltd, 2010.

[12] M. Najmi, A. Zabidi, M. A. Maarof, and A. Zainal. Challenges in high accuracy of malware detection. In *2012 IEEE Control and System Graduate Research Colloquium (ICSGRC 2012)*, pages 123–125. IEEE, 2012.

[13] D. Rentz. *Microsoft Compound Document File Format*. OpenOffice.org, 2007.

[14] D. D. Ruscio, P. Pelliccione, A. Pierantonio, and S. Zacchiroli. Towards maintainer script modernization in foss distributions. In *International Workshop on Open Component Ecosystems 2009 (IWOCE 2009)*, pages 11–20. ACM, 2009.

[15] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 61–72. ACM, 2010.

[16] A. Shabtai, Y. Fledel, and Y. Elovici. Automated static code analysis for classifying android applications using machine learning. In *2010 International Conference on Computational Intelligence and Security*, pages 329–333. IEEE, 2010.

[17] H. Vegge, F. M. Halvorsen, R. W. Nergard, M. G. Jaatun, and J. Jensen. Where only fools dare to tread: An empirical study on the prevalence of zero-day malware. In *2009 Fourth International Conference on Internet Monitoring and Protection*, pages 66–71. IEEE, 2009.

[18] S. Wu. Observations and lessons learned from point-in-time cleaning against real-time protection. In *Virus Bulletin Conference September 2010*, pages 165–170. Virus Bulletin Ltd, 2010.