# RUNTIME POTENTIAL UPDATER FILE(S) IDENTIFICATION:

# DOES YOUR SOFTWARE UPDATES AUTOMATICALLY WITH APPLICATION WHITELISTING?

Janardhan Reddy, Amit Kumar Jha and Sandeep Romana

Centre for Development of Advanced Computing, Hyderabad, India

***ABSTRACT***

*One of the major hurdles to widespread usage of application whitelisting with today's dynamically changing and updating software's; is the static environment it creates that leaves little scope for changes to the system once whitelisting is enforced. The de-facto method to allow for trusted changes to system is to make selected executable files as trusted and allow changes made to the system through these files even when whitelisting is enforced. The problem with this is; difficulty the user faces in identifying the updater files for third party software. In this paper, we present the method to identify the potential updater files for the third party software in a Microsoft Windows environment. Further we test the method for commonly used third party software, presenting the results of experimentation and effectiveness of our approach.*

***KEYWORDS***

*Application Whitelisting, Updater Identification*

## 1.INTRODUCTION

Application whitelisting allows executing approved executable files on a system contrary to the blacklist based approach followed by traditional COTS AV scanners. Hence, whitelisting has inherent characteristic of stopping execution of all unapproved executable files including malware files. Whitelisting technology having the above mentioned powerful advantage is less preferred with dynamic software environments as there is minimal scope for making changes to the system [1]. Whitelisting software is switched to '*installation mode*' for software updating and new installations. Improvement to this approach is to trust the files as updater and then record the changes made by these trusted updater(s) and automatically whitelist the changes. Today whitelisting software's have inbuilt support for common third party software where specific files belonging to software are identified for automatic updating of software. These file are identified at the software provider's end [2]. These files when trusted for updating, records the changes made to file system and updates the whitelist database accordingly. The problem with this approach is in difficulty of process of identifying these software specific updater file(s). To extend the usage of application whitelisting to dynamically changing environments continuous updates to whitelist are required. In this paper, we present a method by which potential updater file(s) belonging to third party software can be identified in a whitelisting environment at the runtime.

## 2. APPROACH

Before we describe the method of updater file identification at runtime, the next subsection explains how handling the process of automatic software updating in our implementation of application whitelisting.

### 2.1. Automatic Updating

To allow of automatic software updating we identify updater files on per software basis and make them trusted updater. These trusted updater files can create further child processes (which in turn will be treated as trusted updaters) until updating finishes. At the time of process creation we extract process ID, parent process ID and if the process is trusted updater (either directly or by being child of trusted updater), add PID of the process to separate list called trusted PID (tPID) list. At the time of process termination we remove the PID from tPID list. The files downloaded by trusted updaters are logged to another database called updates database (updt-db). Apart from verifying the whitelisted files from executable file database we allow files to create process even if they are found in updates database (updt-db). We trust updates database because it contains files downloaded by trusted updater(s). To reflect the changes made to system files during automatic updating we move the files from updates database (updt-db) to system wide executable file database as whitelisted files. Figure 1, lists various data structures used during automatic updating.


Figure 1. Data structures used in Automatic Updating

### 2.2. Potential Updater file(s) identification

To identify potential updater file(s) we collect system wide process creation and file written activity in the logs. Figure 2, describes the data structure of these logs. Along with this log data, information from system wide executable file database is used. These potential updater file(s) can be made as trusted updaters to allow for automatic updating of software.

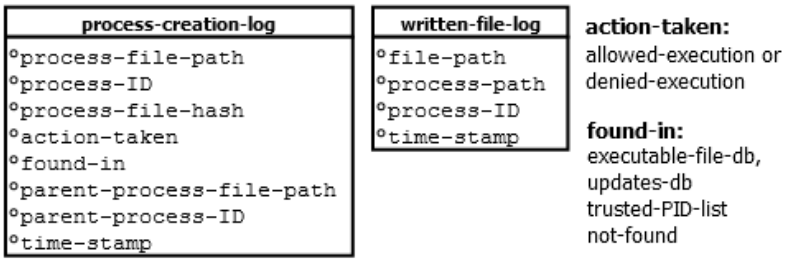



Figure 2. Data structures used in potential updater file identification

Whenever a file is executed it's matched against the data in whitelist and is denied execution when either it is blacklisted or it is not found in the database. The file(s) that are not found in the database are the files that are newly written to the system. We initiate potential updater file

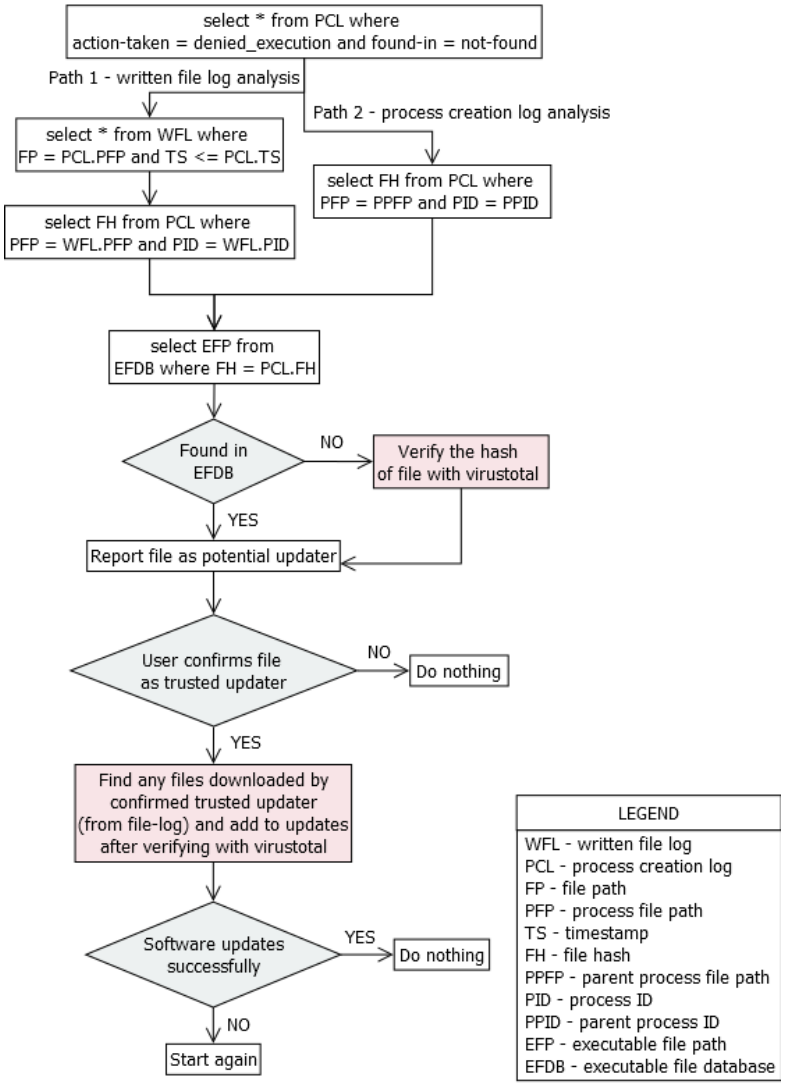
identification with the file that is denied execution because of not found in the database. The method takes two different paths to identify potential updater file. Path 1 uses written file(s) log information and path 2 uses process creation information. The system wide executable file database is used to build a level of trust on identified potential updater file. If the reported file exists in this database, full trust is assigned to it, else the file is considered for further verification. For further verification we propose to use online services such as VirusTotal to confirm its genuineness [3].

Another parameter we consider during the process of updater identification is timestamp. Timestamp helps us to narrow down the number of logs for which we search and further helps to eliminate duplicates and errors that may arise due to multiple executions of same file(s). Each time a not-found file is denied execution, it must have been written to file system with the timestamp of time before it attempted to execute. After the identified potential updater files is trusted as trusted updater (by user or automatically) we identify all the files written by this file and adds to updates database.

Again, to verify the genuineness of the file(s) before adding to updates database we depend on VirusTotal. Finally, while listing the identified potential updater file(s) we filter out windows process such as *msiexec.exe*, *explorer.exe*, *svchost.exe*, *services.exe*, *dllhost.exe* etc. Sequence of steps to identify potential updater file (in pseudo SQL) are as follows:

1. Start with file denied execution and not found in executable file database.
   *Path 1:* Written file log analysis:
2. Select all fields from written file log when file path equals process file path in process creation log and timestamp is less than or equal to timestamp of file selected from process creation log in step 1.
3. Select file hash from process creation log when process file path equals process file path in written file log and process ID equals process ID in written file log.
   *Path 2:* Process log analysis
4. Select file hash from process creation log when process file path equals parent process file path and process ID equals parent process ID.
   *Common Steps:* For both Path 1 and Path 2.
5. Verify of the hash of the file in system wide executable file database. If file is found, list it as potential updater file else scan it with VirusTotal before listing it as potential updater file.
6. Filter out names of common windows processes.

Once the potential updater file is identified we make it as trusted updater, add the files downloaded by that process from the written file log to executable file database as whitelisted after verifying with VirusTotal. Process of potential updater identification along with automatic updating is given in Flowchart 1.




Flowchart 1: Potential Updater Identification and Automatic Updating

## 3. EVALUATION

We evaluated the method presented in this paper with some of the most common windows software's. The test results will all the software's chosen for testing were promising. The test results are presented in the table 1 below. In our observation, for some software's, multiple iterations of identification of potential updater are required to get updated successfully. The example of one such software is notepad++ for which details are provided in table 1 and the case is elaborated below.

*The case of notepad++:* With *notepad++*, the *npp.6.6.9.Installer.exe* was denied execution and *GUP.exe* was identified as potential updater in first iteration. Once *GUP.exe* was made trusted updater, further *notepad++.exe* was denied execution and *npp.6.6.9.Installer.exe* was notified as potential updater. Second time, after making *npp.6.6.9.Installer.exe* as trusted updater the notepad++ updated successfully.

Table 1. Potential updater(s) for common software*

| Software | Version | Iteration | File Denied Execution | Potential Updater with method 1 – file log analysis | Potential Updater with method 2 – process log analysis | Final Potential updater(s) |
|---|---|---|---|---|---|---|
| Adobe Reader | 11.0.03.37 | Iteration1 | AdobeARMHelper.exe | AdobeARM.exe | AdobeARM.exe | AdobeARM.exe |
| Avast | 9.0.2013.292 | Iteration1 | instup.exe | instup.exe | instup.exe | |
| | | | instup.exe | instup.exe | AvastSvc.exe | instup.exe, AvastSvc.exe |
| AVG | 10.0.0.0 | Iteration1 | avgmfapx.exe | avgmfapx.exe | avgmfapx.exe | avgmfapx.exe |
| Firefox | 27.0.1 | Iteration1 | helper.exe | updater.exe | updater.exe | |
| | | | maintenanceservice_installer.exe | updater.exe | updater.exe | |
| | | | firefox.exe | updater.exe | updater.exe | updater.exe |
| ITunes | 11.1.4.62 | Iteration1 | SetupAdmin.exe | SoftwareUpdate.exe | dllhost.exe | SoftwareUpdate.exe |
| Java Development Kit | 7.0.110.21 | Iteration1 | jre-7u67-windows-i586iftw.exe | -- | jucheck.exe | jucheck.exe |
| Java Runtime Environment | 7.0.110.21 | Iteration1 | jre-7u67-windows-i586iftw.exe | -- | jucheck.exe | jucheck.exe |
| Notepad++ | 6.6.4 | Iteration1 | npp.6.6.9.Installer.exe | GUP.exe | GUP.exe | |
| | | Iteration2 | notepad++.exe | npp.6.6.9.Installer.exe | npp.6.6.9.Installer.exe | GUP.exe |
| Picasa | 3.8.0 | Iteration1 | PicasaUpdater_3073.exe | setup.exe | GoogleUpdaterService.exe | |
| | | | PicasaUpdater_3073.exe | setup.exe | setup.exe | GoogleUpdaterService.exe, setup.exe |
| Safari | 5.34.52.7 | Iteration1 | SetupAdmin.exe | SoftwareUpdate.exe | dllhost.exe | SoftwareUpdate.exe |
| Team-viewer | 4.0 | Iteration1 | update.exe | TeamViewer.exe | TeamViewer.exe | TeamViewer.exe |
| Thunderbird | 4.42 (24.1.0) | Iteration1 | helper.exe | updater.exe | updater.exe | |
| | | | maintenanceservice_installer.exe | updater.exe | updater.exe | updater.exe |
| TrendMicro | 3.0 | Iteration1 | post-load.exe | coreServiceShell.exe | coreServiceShell.exe | |
| | | | uiWatchDog.exe | coreServiceShell.exe | coreServiceShell.exe | |
| | | | uiWinMgr.exe | coreServiceShell.exe | explorer.exe | |
| | | | TmExtIns.exe | coreServiceShell.exe | coreServiceShell.exe | coreServiceShell.exe |

* The name for the file that is denied execution and the file reported as potential updater may be same but their hashes were different, so they are not the same files.

## 3. CONCLUSIONS

In this paper we presented a method to identify potential updater files for third party software at runtime which is capable of replacing the currently used non-user friendly way of doing the same either by the solution provider and/or the user of the software.

One disadvantage of the method is that malware files may exhibit behaviour similar to potential updater file(s). For this we crosscheck with VirusTotal database which is a dependency. In absence of such kind of service solution provider has to have his own database of malware file hashes.

With runtime identification of updater files combined with automatic updating of software when whitelisting is enforced helps to dynamically handle changes to whitelist during automatic

updating. This definitely is a more user friendly approach which will help dynamically changing environments to take advantage of application whitelisting.

## ACKNOWLEDGEMENTS

We would like to extend our thanks to anonymous reviewers you contributed to improving this paper.

## REFERENCES

[1] Eswari, P.R.L.; Babu, N.S.C., "A practical business security framework to combat malware threat," Internet Security (WorldCIS), 2012 World Congress on, vol., no., pp.77,80, 10-12 June 2012.

[2] Christopher Gates, Ninghui Li, Jing Chen, and Robert Proctor. 2012. CodeShield: towards personalized application whitelisting. In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12). ACM, New York, NY, USA, 279-288.

[3] VirusTotal - Free Online Virus, Malware and URL Scanner, www.virustotal.com

**Authors**

Janardhan Reddy working with C-DAC Hyderabad since March, 2012 as part of E-security team. His area of interest includes application whitelisting, End Point Security, Network Security.

Amit Kumar Jha is currently working in e-Security team at Centre For Development of Advanced Computing (C-DAC) Hyderabad. His area of interest includes anti-rootkit detection technique and application whitelisting.

Sandeep Romana has more than 7 years of experience in the field of research and development of security software for desktop's and small networks. His areas of research include behavioural malware detection and application whitelisting.