International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Novel Real Time Offline Patching Scheme with Secured Logging Over Cloud

Sunita[1], Shridevi Soma[2]

[1]M.Tech. (CSE) Student, Computer Science and Engineering Department, PDA College of Engg., Gulbarga, Karnataka, India

[2]Associate Professor, Computer Science and Engineering Department, PDA College of Engg., Gulbarga, Karnataka, India

Abstract: ***Modern day software systems are updated online through an update server. A metadata file is installed at the update server that exposes the information about current version, build, software changes, system requirement, information about fixes, priority and also a URL to the software bundle offered as update. The software installed in client machine checks for this metadata as and when it is scheduled to check for update and has an internet connection. Once the software finds out that there are updates available, it asks the user for permission to update. In update process it silently downloads the patch or the bundle file and updates the software. Once updated the new version is launched. This technique has certain disadvantages that it is not centralized and is not suitable for patching software in a cloud architecture where several virtual machines collaborates to provide service. The update and patching information is stored locally as a log file which does not provide any security. In this paper we propose a unique patching scheme to overcome the aforementioned drawback of current OPS in which metadata of the image and the patching and update information is stored as a secured file which is encrypted using symmetric key encryption using AES and Rijndael. It is also demonstrated the feasibility of the system on real windows updates by enabling centralized offline updating the windows updates. In the offline scheme, updates and patches are downloaded using a background agent without the need for the software to be running. Once patches are available, they can be scheduled to be applied at a later instance. This provides better bandwidth utilization. Results show that the complexity of the process is O(k+1/n) and that the system does not add any significant overhead for security extension.***

Keywords: Offline patching, Cloud Computing, AES, Secured Log Access

1. Introduction

Software patching, updating and version management are important aspects of software life cycle. Several patching and version management schemes are being proposed. However most of the schemes are online which needs the software to search for a percolated metadata for update when the software is running and triggering the update process. Also such update history is maintained locally in the system as unsecured log file. In modern computing environment where several virtual machines collaborate to provide a computation solution, it is quite difficult to manage such stand alone patching. Also in critical software system, we need more security. Centralized cloud based storage of the log and accessing it thereafter is also important because this provides a centralized control to the patching and version management. Current solution does not meet these needs. Hence it is important to develop system that can provide a centralized secured logging of the patching and offline patching. The objective of this work is to provide a software architecture that enables offline patching of the software as well as a central version/log access mechanism that is secured using symmetric key cryptography. Offline patching means searching for Patch, downloading and applying the patch: all can be done independently of one another. Hence the system does not need to run a specific patch while applying it

The objective is also to demonstrate that the architecture is adaptable for real and high end software solution by extending the system to windows services.

2. Related Work

For patch management a game theoretic model [1] has been developed to derive the optimal frequency of patch updates to balance the operational costs and damage costs associated with security vulnerabilities. Cloud computing security issues and Challenges [2] analyze cloud computing vulnerabilities; security threats cloud computing faces and present the security objective that need to be achieved. In [3] authors present preliminary ideas for the architecture of a flexible, efficient and dependable fully decentralized object store which is able to manage very large sets of variable size objects and to coordinate in place processing. To handle images for different infrastructures spanning virtualized and non-virtualized resources, the FutureGrid [4] user controlled image management framework has been developed as a revolutionary way. It allows users to register images, created by their software, for Nimbus, Eucalyptus, OpenStack, as well as bare-metal instantiations. A generic catalog and image repository [5] has been proposed to store images of any type which is useful for FutureGrid, also for any application that needs to manage IaaS images. Mirage is an image library [6] which stores images in a format that indexes their file system structure instead of as opaque disk images. Like other libraries, Mirage provides features for image capture and deployment. In addition, Mirage maintains a provenance tree that records how each image was derived from other images; allows operations, like patching and scanning, that normally require a VM instance to execute offline; and enables analyses such as image search and comparison. For maintaining the security for VM images in [7] it has been proposed new Cryptographic protocols which take full advantage of the unique properties of public key cryptosystems. To address the risks, that face administrators and users (both image publishers and image

retrievers) of a cloud's image repository [8], the image management system has been proposed that controls access to images, tracks the provenance of images, and provides users and administrators with efficient image filters and scanners that detect and repair security violations. An on-line software replacement system [9] is for on-line software version change for software written in the C language. This system eliminates the shutdown while installing a new version of a piece of software. A novel tool named Nüwa [10] that enables efficient and scalable offline patching of dormant VM images such VM images are often left vulnerable to emerging security threats.

3. Image Security and System Architecture

Software may consist of one or more exe files and several other files and entries which may include dll files, ini files, and registry entries and so on. While a software is distributes, each of these files are important for the functioning of the software. If the files are distributed individually, there are always a chance of deletion of some file and few files getting corrupted, which ultimately leads to undesirable functioning of the software.

Therefore these files are bundled as a single zip/jar/installer file. An image is a standard packaging of the software system just like the installer with added functionalities like booting, auto run if presented through CD/DVD etc. Images come with CRC facility that makes checking the software sanity check easy and efficient. Therefore in the proposed work it has been published/uploaded the software as well as the patch of the software as software image into the cloud.

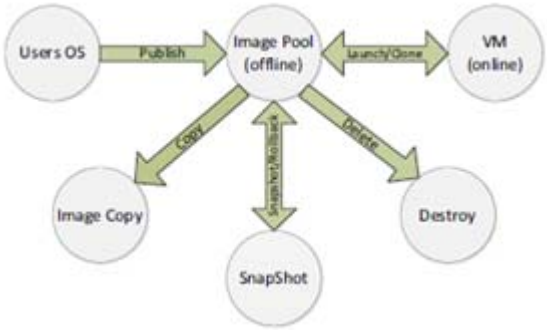

Figure 1: Offline software patching over cloud

A snapshot is the current state of the software running in a virtual machine. As the proposed system supports distributed metadata management and patching, it has been considered virtual cloud architecture as above for analysis of our system.
Developer (users) may upgrade a software version or release a patch. The patch is released as an image pool (another snapshot of software) rather than discrete files. This image pool is stored in the cloud storage and is visible to all the virtual machines in a cloud. In case of a standalone customer machine, one can imagine the stand alone system as a single virtual machine over a cloud comprising of only one computer. The machines looks for patches and updates and when available may download in a secured location.

Downloading files in a separate location for applying the image patch is known as offline patching. An online patching is one where a running program first copies the files from remote locations into its main program memory and then commit the changes when the program is terminated.

Online patching is difficult to rollback. However incase of offline patching, user may download the files, scan them for viruses, check the digital signature and then may prefer to apply the patch. Thus it gives more control over the software patching process.

Once the patch is changed, it needs to be updated in the patch metadata for all the virtual machines in the cloud. Managing independent copies of history for every machine is not feasible on the cloud. Hence it should be centralized. It has been already discussed in Current system how the current system offers difficulty in history and version management of patches in a distributed environment. Therefore to overcome the drawback it has been offered a cloud based solution for version management after the patch is applied. No matter whichever VMs the patch is applied, the information about the machine ID, current version, patching details, patched versions are updated in a central patch file which is located in the skydrive for all time access.

It has been used Rijndael and AES based security for cloud. The entire architecture from publishing the patch to applying the patch and its version management is as presented below.

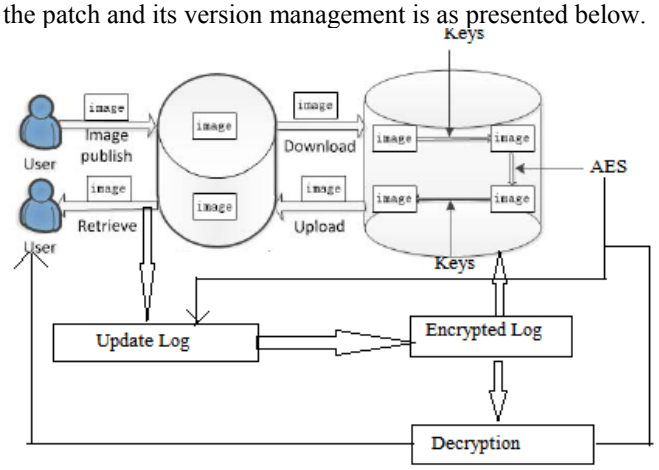

Figure 2: Architecture of the proposed system

4. Methodology

As it can be seen in the block diagram below, first the user downloads particular software from the cloud space and installs in his system. There are three major components called **Collector**, **Uploader** and **Updater**.

Collector is a component which checks for software's version and if the patch is available, it is downloaded and software is updated. If the update process is successful, the information about previous version, current version, machine name, data and time are logged in a log file. This log file is encrypted with a key provided by user. This log file is stored in Skydrive (Public Cloud). User can access the log file any time from the cloud and then decrypt it to view the contents.

Uploader is a component where publisher creates a patch file and uploads in an authenticated web server (Private Cloud) using FTP services. This is invariably the vendor side tool that helps distributing a patch efficiently in distributed environment.

Updater is a component where the patch is applied on software or its version is updated.

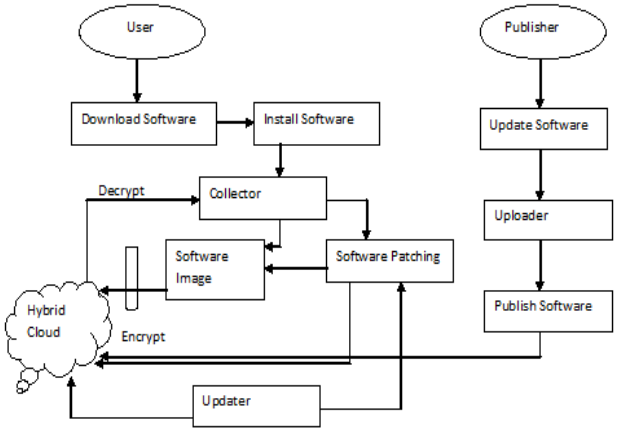

Figure 3: Block Diagram of the System

Implementation

Collector Module

The responsibility of this module is to Store the log locally before encryption. The log file skpath.txt will be present in the relative folder of /User/AppData/Roaming/../skpath and is actually the file which stores the location of the skydrive log file.

Getting the information about the current version of the software such that the version can be compared with the software version published and if new version is available as patch can be downloaded. This is performed by using reflection technique of the .Net assemblies. It will ask the user for his skydrive credential and if the credential is correct, download the file from skydrive location specified by the skpath.txt. Update the data in the datagrid. Separate each column and update the software path.

Now obtain the update metadata from remote location. If the metadata is not null, i.e. software patch is available, compare the current version of the patch with the published patch version. If the current version is lesser than the patch version then only trigger the patching/ updating process. Following image shows the patch in the real server.

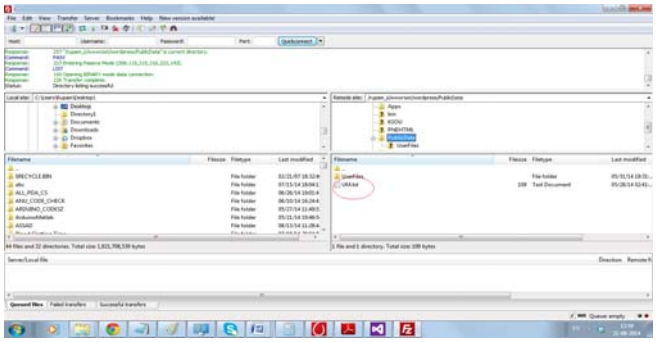

Figure 4: Real server showing the patch

The content of the UM file is as shown below.

Figure 5: Content of UM file

As it can be seen from the diagram that there are four parts of the file: The name of the application image, its version information, release date and download url respectively. Download URL is the absolute path where the UM.txt file is located. It can be seen clearly in the FileZilla image that UM.txt is located in Public Data folder. Once the information is available, the image can be downloaded and patch can be applied. This can be done by a method that enables downloading of the image file into a location using asynchronous web downloading technique. The image is downloaded as a zip file. If the downloading was successful, then it unzips the software and triggers install.

Encryption using AES block cipher algorithm which is like a service. The content of the log file is passed as argument to the encryption service as string along with a key for encryption. The result of Encryption is stored in the Skydrive. Encrypt function takes a random initial vector and mixes the password with that for more security. Both message and mixed passwords are converted to byte array. Then Symmetric Rijndael encryption (AES) is used to encrypt the message. Once encrypted, message is converted back into string and returned.

Windows Update Service Module

This module checks if any update is being released by Microsoft for the underneath operating system. Updates could be patches, fixes, or feature updates. Here we would consider all the types as patches only. The responsibility of this module is to search for those updates which are not yet installed in a specific virtual machine, download the updates silently and allow installation of the update. Further the system supports a log management system which is similar to that of custom and independent software log management system. It has been created an update session by a method which is first step to create a secured connection between the virtual machine and the Windows Update server. It then searches for those patches/updates that are not installed in current system or that are not present. Once the search result is available, it is enumerated in a list and displayed in a list box. The following figure shows Windows services showing the available installs and installed patches from skydrive.

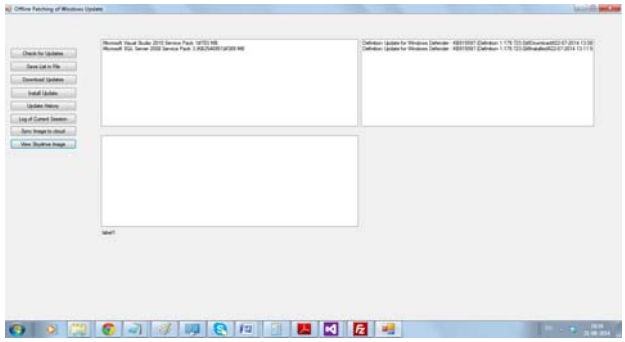

Figure 6: Windows services showing the available installs and installed patches from skydrive

5. Result and Experiment Analysis

In this work first a software model is created that enables offline patching and central cloud based secured logging of the patching history. Custom software is designed in C#.Net. Software is created for releasing the software and its patches. This release tool automatically detects the version of the software and then creates a metadata that describes the version. Further this release tool bundles the metadata and the patch as a single zip file and uploads in an authenticated web server using FTP services.

Once the customer installs software from dropbox location, a background process checks for available patches in the metadata created by the release tool. If the patch is available, it is downloaded and software is updates. If the update process is successful, the information about previous version, current version, machine name, data and time are logged in a log file. This log file is encrypted with a key provided by user. This log file is stored in Skydrive. User can access the logfile any time from the cloud and then decrypt it to view the contents. Even if the system is restored or OS level changes are incurred, the patching information remains safe due to their availability in cloud unlike present system where system updates causes a loss of such log files.

In order to demonstrate the adaptability of the system, the architecture has been extended for windows update services and showed that the proposed system can be flawlessly adapted and extended by any software patching.
The proposed system helps in maintaining a patch history in most innovative way by storing it securely over the cloud. Therefore the metadata can be obtained and synced with any number of domain computers or group of virtual machines. The pair wise cryptography does not require any key exchange mechanism and hence is quite light weight.

Hence the proposed system provides a much better technique for updating/ patching the software and log management of the same. The offline patching process allows the user to explicitly download the patch rather than applying it in-memory when the system is running. Hence the system is much more efficient. When the method is extended to windows patch, still the system functions in a smooth way. That proves that the proposed system can be integrated under any software patching and update management efficiently without losing accuracy.

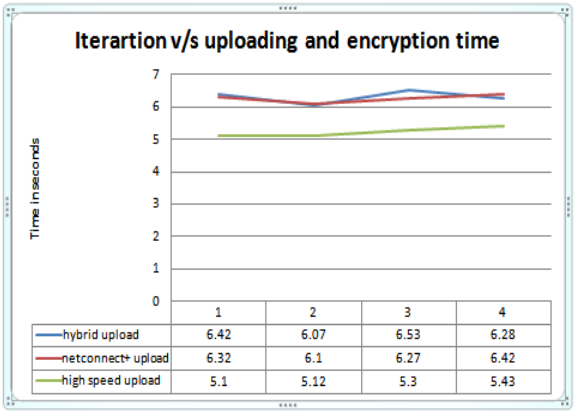

Figure 7: Iteration v/s Uploading time and encryption time

This graph shows that the security extension and Skydrive integration does not add too much overhead on the patching time. We test the system under three different internet speed settings. As expected for high speed broadband internet the patching process is quicker. However in other two settings the patching process takes moderate time under 10 seconds. In all the iterations, size of the file is kept almost constant. Uploading time here refers to the time required by the VM to upload/update the log file plus the time taken for encrypting the data of the file.

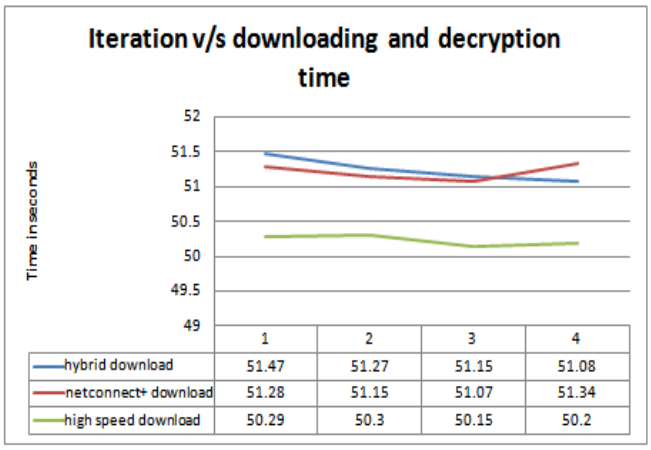

Figure 8: Iteration v/s downloading and decryption time

Before the log file is updated, it needs to be downloaded from the skydrive, decrypted, and then current content can be appended with the data. We check the time taken for downloading and decrypting the log file from the skydrive. It is important to note that as the software iteration increases, entry of the file also increases. Hence the complexity can be ideally thought of O(n) where n is the number of patching. However it can be seen that the time does not increase linearly with increase of time. Hence the complexity is O(k+1/n) where k is a constant that depends upon internet characteristic.

6. Conclusion and Future Work

Submission Software patching is an essential aspect of software version management and software bug fixing. The process can be both online as well as offline. Offline patching schemes are more popular over the other one because of its advantages like more control of user over the patching decision and an efficient version management.

With the advancement of cloud computing and its increasing popularity, it is essential to offer a secured patching architecture with centralized version control that should keep track of each virtual machines in the system. The proposed architecture has two major contribution: firstly it helps to conceive the client as a cloud system and offers the patches that can be independently applied to all the virtual machines, secondly it offers a real time secured logging that helps the update history to be stored in cloud with encryption. It has been used a symmetric key encryption with AES to reduce the key exchange overhead.

Results show that the system can be adapted to real time and more heavy software architecture like windows update. Performance analysis shows that for a moderate internet speed the proposed system adds slight overhead of about 15seconds for the patches. Considering the security and data protection that the proposed system offers, this is one of the small trades off in the system.

As a future work the system can be extended to support software patch/update rollback system. Also in a cloud environment, secured paired key cryptography with attributes like CP-ABE can be used for stronger security.

References

[1] Cavusoglu, Huseyin, Hasan Cavusoglu, Jun Zhang. "Economics of security patch management." The fifth workshop on the economics of information security (WEIS 2006). 2006.
[2] Krešimir Popović, Željko Hocenski “Cloud computing security issuesand challenges” In MIPRO, 2010 Proceedings of the 33rd Internatjional Convention pages 344-349, Osijek, Croatia
[3] Ricardo vilaca, Rui oliveira “Clouder: A Flexible Large Scale Decentralized Object Store. Architecture Overview.” Proceeding of WDDDM ’09,pages 203-210
[4] Javier, Diaz, Gregor von Laszewski, Fugang Wang, Geoffrey Fox."Abstract image management and universal image registration for cloud and hpc infrastructures." Cloud Computing(CLOUD),2012 IEEE 5th International Conference on. IEEE, 2012.
[5] J. Diaz, G. von Laszewski, F. Wang, A. Younge, and G. Fox, “FutureGrid Image Repository: A Generic Catalog and Storage System for Heterogeneous Virtual Machine Images,” Third IEEE International Conference on Coud Computing Technology and Science (CloudCom2011), 2011
[6] Glenn Ammons, Vasanth Bala, Todd Mummert, Darrell Reimer, Xiaolan Zhang. "Virtual machine images as structured data: The mirage image library." Usenix HotCloud (2011).
[7] Merkle, R. C. “Protocols for public key cryptosystems.” In Proceedings of the IEEE Symposium on Security and Privacy (1980), pp. 122–133.
[8] WEI, J., ZHANG, X., AMMONS, G., BALA, V., AND NING, P. ” Managing security of virtual machine images in a cloud environment.” In Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW ’09) (2009), pp. 91–96.
[9] Wu Zhou, Peng Ning, Xiaolan Zhang, Glenn Ammons, Ruowen Wang, Vasanth Bala. "Always up-to-date: scalable offline patching of VM images in a compute cloud." Proceedings of the 26th Annual Computer Security Applications Conference. ACM, 2010.pages 23-30
[10] Deepak Gupta and Pankaj Jalote.“On line software version change using state transfer between processes.” *Softw. Pract. Exper.*, 23(9):949–964, 1993.